An Assessment of Machine Learning Approaches for Predicting the History-Dependent Deformation of Dual Phase Steels

Submitted in partial fulfillment of the requirements
of the degree of

Dual Degree (B.Tech. + M.Tech.)

by

Sarthak Khandelwal
(Roll No. 16D110006)

Supervisors:
Prof. Anirban Patra

Metallurgical Engineering and Material Science
INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
2021

Certificate

It is certified that the work contained in this thesis entitled ”An Assessment of Machine Learning Approaches for Predicting the Time-Dependent Deformation of Dual Phase Steels” by ”Sarthak Khandelwal” has been carried out under my supervision and that it has not been submitted elsewhere for a degree.

Prof. Anirban Patra

July 2021 Metallurgical Engineering and Material Science

Indian Institute of Technology Bombay

Abstract

Name of the student: **Sarthak Khandelwal** Roll No: **16D110006**

Degree for which submitted: **Dual Degree (B.Tech. + M.Tech.)**

Department: **Metallurgical Engineering and Material Science**

Thesis title: **An Assessment of Machine Learning Approaches for Predicting the History-Dependent Deformation of Dual Phase Steels**

Thesis supervisor: **Prof. Anirban Patra**

Month and year of thesis submission: **July 2021**

Surrogate machine learning models are proposed for modeling the elastic-plastic deformation of dual phase microstructures. The deformation is first simulated using a J2 plasticity model, whose results form the basis for surrogate model development. A simple Artificial Neural Network (ANN) model is applied to make the predictions for von-Mises effective stress, stress triaxiality ratio and effective strain. We further test the performance for a Long Short Term Memory (LSTM) based Recurrent Neural Network (RNN) and compare the results obtained from the two models. For both the approaches, multiple models were trained and the one, with least overall error, was used to make the predictions. There is a significant increase in the accuracy with the latter model, which has also been used to predict the deformation contours in untrained microstructures, as well the aggregate stress-strain response and the strain partitioning between the two phases.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Finite Element (FE) modeling frameworks have been used extensively to simulate the spatio-temporal deformation (and evolution) of microstructures. Such frameworks are based on the development of constitutive equations to represent the underlying deformation mechanisms. These frameworks have generally been developed, with various homogenization approaches (if required), to study the deformation at the level of components [3, 4], microstructures [5, 6], and nano-scale structures [7, 8, 9]. The FE frameworks often employ physically-representative Crystal Plasticity (CP) constitutive equations [5, 10, 11] and phase field modeling approaches [12, 13, 14, 15] to study the microstructure evolution during deformation and correlate them with the overall mechanical response. Additionally, Fast-Fourier Transform (FFT) based spectral solvers [16, 17, 18] have also been developed to study the deformation of microstructure ensembles. These predictions are also compared with relevant experimental measurements [19, 20, 6], where available. These spatially resolved deformation simulations are generally considered to be the state-of-the-art and provide high fidelity information regarding the local deformation characteristics. They have become the tool of choice for studying the effect of microstructure on the deformation behavior. However, these methods generally have large associated computational costs, which may act as a deterrent for performing large scale simulations for advanced computational materials design problems.

In recent years, surrogate modeling approaches are being explored as alternative approaches that can significantly reduce the computational times, while maintaining the high fidelity of microstructural information obtained from FE simulations. These approaches generally employ a combination of data informatics, image-based modeling and neural networks to predict the microstructure characteristics during deformation [21, 22, 23]. This is accomplished by training the surrogate models to the spatio-temporal data obtained from the FE or FFT simulations [24, 25, 26, 27, 28, 29, 30]. In some cases, the FE simulations may even be bypassed by directly training the surrogate models to the experimentally obtained microstructure evolution data [1]. Further, surrogate constitutive model forms have also been trained using data-based approaches to the "original" model response [31, 32, 33] and implemented in FE frameworks to obtain reduced computation times [32, 34]. Such surrogate models broadly fall in the domain of machine learning techniques.

Machine Learning (ML) is an emerging research field and has been widely adopted in recent years due to its ability to predict properties, relationships and inferences from different kinds of data, with relatively lower computational costs [35]. While research in this field has increased significantly in the recent years [36], we mention a few examples of the application of ML techniques specific to mechanics and microstructure evolution problems in the following (this being the central theme of our work). Mangal and Holm [27] used ML to predict the formation of stress hot spots in face centered cubic materials. They implemented a random forest algorithm, which was used for classification of grains as stress hot spots, where localized deformation may occur. The algorithm also successfully captured the effect of changing material and texture parameters on the stress hot spots. Muhammad et al. [1] implemented an Artificial Neural Network (ANN) model to predict the local strain distribution, fracture, and evolution of plastic anisotropy in additively manufactured alloys. They were able to predict the location, intensity and the shape of shear bands before failure with reasonable accuracy. An image-based Convolutional Neural Network (CNN) model was developed by Beniwal et al. [23] for predictive modeling of structure-property linkages. In their work, the reduced order model has the ability to predict stress localization simply from the microstructure image. Machine learning was also used to homogenize arbitrary microstructures and obtain the corresponding nonlinear, history-dependent constitutive response [34]. A Long-Short Term Memory (LSTM) based RNN network has the capability of remembering states over time and thus capable of modeling dynamic systems [37]. Pandey et al. [29] developed a LSTM-based framework for predicting the texture evolution under tensile loading of polycrystalline materials. They used a first-order approximation by taking the effect of only nearest neighbor interactions on each material point. Although their model did not account for any long range interactions, it produced results with 99% accuracy. Moreover, the prediction capability of their model for different microstructures was not explored exhaustively.

Most of the above mentioned surrogate modeling studies generally focused on predicting the elastic-plastic deformation of single phase microstructures. Multi-phase microstructures are generally associated with heterogeneous deformation characteristics and partitioning of stress and strain in different regions of the microstructure [19]. This poses a challenge for ML-based surrogate modeling approaches, as the evolution stress and strain distributions is not only history-dependent, but also dependent on the spatial heterogeneity of the underlying microstructures. Further, while different ML models have been used for solving mechanics problems, their efficacy for simultaneous prediction of the spatio-temporal evolution of the stress and strain distributions in such microstructures has not been systematically studied. This is particularly important given that plastic deformation, the dominant deformation mechanism in metallic systems, is a history-dependent process.

In the present work, we focus on predicting the history-dependent microstructure evolution of dual phase microstructures during deformation using surrogate ML models. Representative dual phase microstructures, comprised of a hard phase and a soft phase, are first simulated using a dislocation density-based, J2 plasticity FE model. The output from the FE model is used to generate the data set for surrogate model training and also serves as the ground truth data. A systematic study of different LSTM-based surrogate models, and trained with the same FE data set, is performed for predicting the deformed microstructure evolution. Specifically, effective strain, von Mises effective stress, and Stress Triaxiality ratio (ST) are chosen as the output variables of the models. The model with the least error in predictions is then used for predicting the spatial deformation contours, aggregate stress-strain response as well the strain partitioning for untrained microstructures.

Figure 1.1 shows the classification of some machine learning algorithms commonly used in materials science.

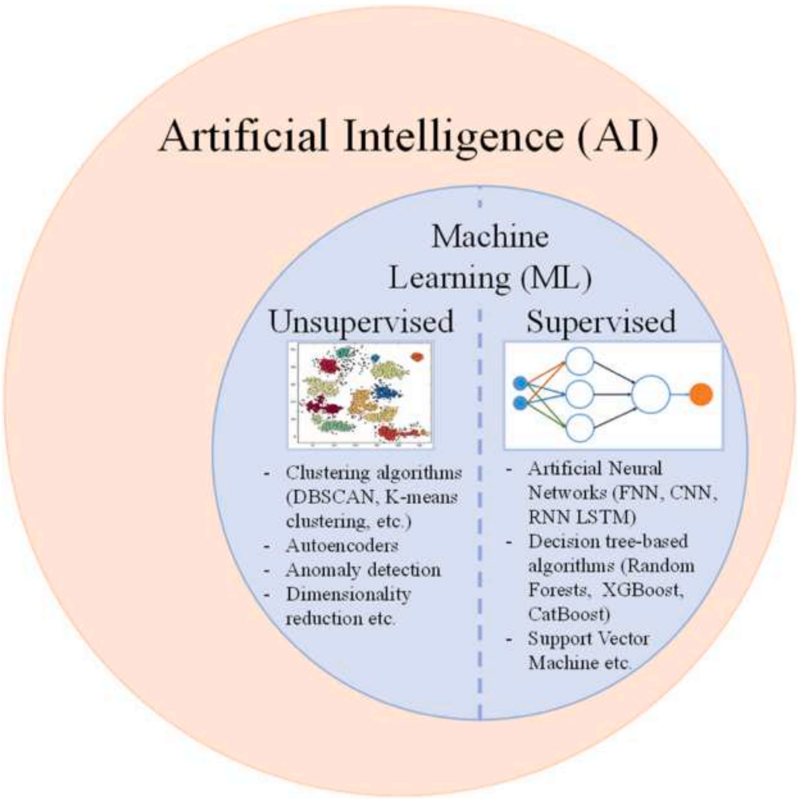

Artificial Intelligence (AI)
Machine Learning (ML)
Unsupervised
Supervised
- Clustering algorithms (DBSCAN, K-means clustering, etc.)
- Autoencoders
- Anomaly detection
- Dimensionality reduction etc.
- Artificial Neural Networks (FNN, CNN, RNN LSTM)
- Decision tree-based algorithms (Random Forests, XGBoost, CatBoost)
- Support Vector Machine etc.

FIGURE 1.1: Various types of machine learning models [1].

1.2 Plan Of Thesis

A systematic study of various deep learning techniques for predicting the evolution of the deformed microstructure a dual phase (DP) steel has been performed in this thesis. A dislocation density based J2 plasticity finite element framework is used to run deformation simulations of the two phase microstructures of dual phase steels. The J2 plasticity simulations are considered as the ground truth for developing the machine learning model. We start with an ANN and optimize the hyper parameters to achieve the best possible results. Further, as plastic deformation is a history-dependent process, LSTMs are better suited for modeling them as they have the capability to remember states over time. Hence, in the next approaches we show a vanilla LSTM and develop on it to improve our results. The data we are dealing with is spatio-temporal in nature. We demonstrate the ability of LSTM to deal with this kind of data and produce accurate results.

The subject matter of the thesis is presented in the following 5 chapters:

- Chapter 2 discusses the fundamentals of deep learning, including the underlying theory of the algorithms used in our work. Further, we discuss some of the approaches used in the literature and their shortcomings.
- Chapter 3 first provides a summary of the J2 plasticity model used for DP steels. We then discuss the ML model development and implementation.
- Chapter 4 discusses preliminary results obtained from our model.
- Chapter 5 summarizes the work done so far and provides details of work planned in the next phase.

Chapter 2

Review Of Literature

2.1 Introduction To Machine Learning

Machine Learning (ML) is the sub-field of the much broader concept of Artificial Intelligence. The primary goal of ML, applied to the problem of interest here, is to understand and develop correlations within a given data, which improve automatically through experience. Specifically, these correlations (or mathematical models) are not prescribed *a priori* and are developed by the ML algorithms via analysis of the input data. In classical computing, most algorithms are sets of programmed instructions which the computers use to calculate or solve the problem. ML algorithms on the other hand enable a computer to train on input data and output values that lie within an acceptable range using statistical analysis. Machine learning finds extensive use in many present day technologies. Some real world applications are shown in Figure 2.1.

Deep Learning is a branch of machine learning which is completely based on artificial neural networks. An artificial neural network (ANN) is a flexible mathematical structure which is capable of identifying complex nonlinear relationships between input and output data sets. It achieves great flexibility and power by learning to represent the world as a nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representation computed in terms of less abstract ones [38]. A number of

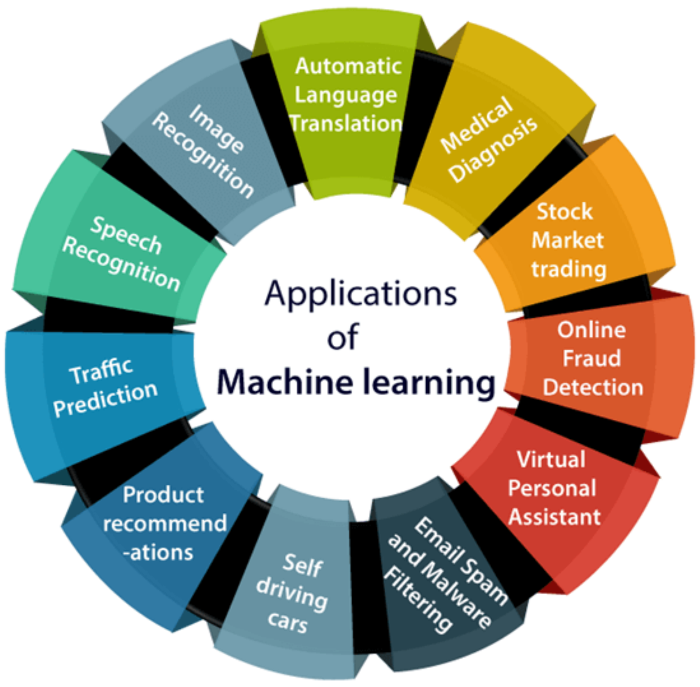

Applications of Machine learning
Automatic Language Translation
Medical Diagnosis
Stock Market trading
Online Fraud Detection
Virtual Personal Assistant
Email Spam and Malware Filtering
Self driving cars
Product recommend -ations
Traffic Prediction
Speech Recognition
Image Recognition

FIGURE 2.1: Various applications of machine learning [2].

deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing and audio recognition. These networks have produced results comparable to and in some cases surpassing human expertise. In this thesis, we have employed deep neural networks and recurrent neural networks to develop a model for predicting the deformation behavior.

2.2 Deep Learning

In this section, we describe the fundamentals of neural networks.

2.2.1 The Neuron

A neuron is a fundamental unit of the human brain. The core functionality of a neuron is to input information from other neurons, process it and pass on the results to other cells. It receives information from different connections, which are dynamically strengthened or weakened based on how often it is used (learning new concepts) and this strength determines the effect of the input on the output. Once the inputs are weighted by the strength of their connections, they are summed and transferred to the next neuron. This concept of a neuron can be easily represented on a computer. An artificial neuron has the capability to take in inputs, $x_i, i \in 1, N$ (for N neurons), which are multiplied by their respective weights, w_i, and then summed to produce the output or logit of that neuron which can be given as:

$$z = \sum_{i=1}^{i=n} w_i x_i \tag{2.1}$$

A constant may be added to this logit and then passed through a function f to produce the output $y = f(z)$. The function, f, is called the activation function and introduces non-linear behavior in the neuron. Alternatively, the logic may be expressed as a dot product between two vectors, $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ as the input vector and $\boldsymbol{w} = [w_1, w_2, ..., w_n]$ [39]. The expression thus becomes:

$$y = f(\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{2.2}$$

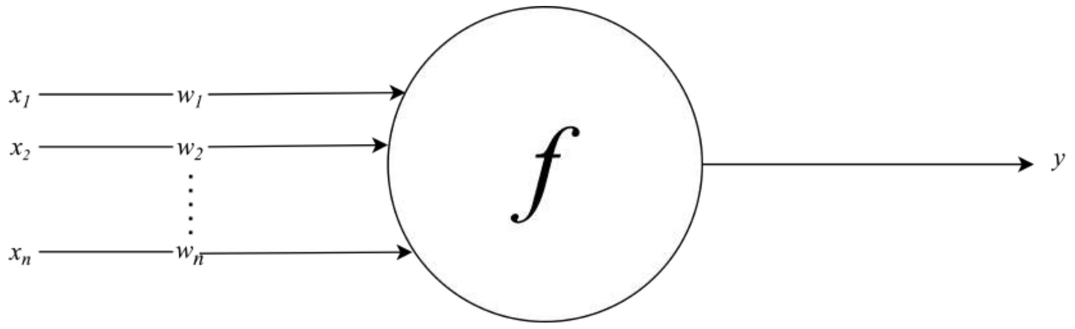


FIGURE 2.2: Schematic of a neuron.

2.2.2 Artificial Neural Network

A single neuron may not be sufficient to solve or model complicated problems. Analogous to our brain, which is made up of millions of neurons arranged in multiple layers, we can construct an artificial neural network. A neural network consists of a number of neuron hooked to each other. The first layer of nodes receives the input data (input layer) and the last layer's output (output layer) is the final answer computed by the neural network. All the layers in between are called the hidden layers. The weight of the connection between the i^{th} neuron in the k^{th} layer with j^{th} neuron in the $(k+1)^{th}$ layer is represented by $w_{i,j}^{(k)}$. Our network's ability to solve problems, is dependent upon finding an optimal value of these weights [39]. Some important aspects of neural networks are as follows:

1. Neurons in the same layer have no connections between them and no data is transmitted from a higher layer to a lower layer.
2. It is not necessary for all the layers to have the same number of neurons.
3. All the neurons may not have their outputs connected to all the neurons in the next layer. Some connections are dropped to optimize the network and the training process.
4. The input and outputs are vectorized representations.

Figure 2.3 shows a simple neural network with 3 layers (input layer, 1 hidden layer and output layer) and 3 neurons in each layer. Mathematically, neural networks can be represented as a series of vector and matrix multiplications. Let us say $\boldsymbol{x} = [x_1, x_2, ..., x_n]$ is the input to the i^{th} layer, which propagates the output $\boldsymbol{y} = [y_1, y_2, ..., y_m]$. We can represent the weights of connections in the form of a weight matrix $\boldsymbol{W}$ of size $m \times n$ along with a bias vector $\boldsymbol{b}$. Then the following holds true:

$$\boldsymbol{y} = f(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b}) \tag{2.3}$$

A single neural network can consist of different neurons stacked together in different layers. The activation function, f, a neuron applied to their logit, z, decides the type of neuron it is. For example, a linear neuron may be of the form: $f(z) = az + b$. A linear neuron is easily computed, but runs into a number of limitations. A network with only linear neurons can mathematically be expressed as a network without hidden layers [39]. Therefore, in

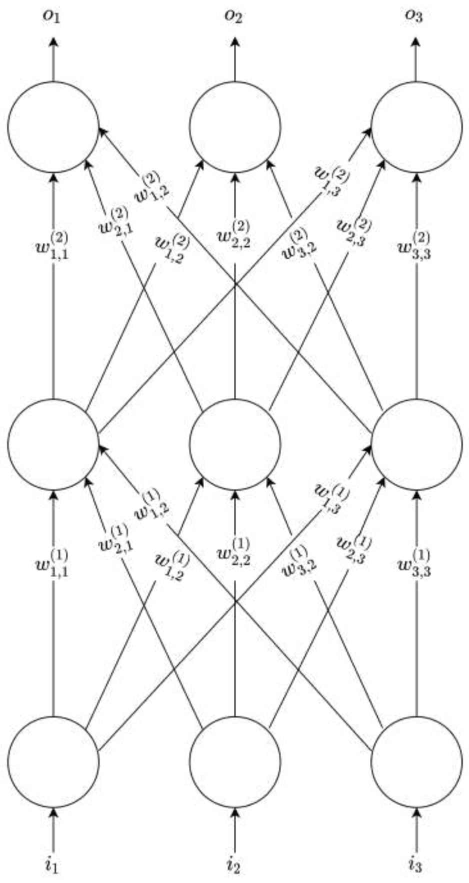


Figure 2.3: Schematic of a simple neural network.

order to learn complex relationships we need an activation function which can employ some non-linearity. To help achieve this, there exist 3 major types of neurons. The first one is the sigmoid neuron:

$$f(z) = \frac{1}{1+e^{-z}} \tag{2.4}$$

From the equation above, we can see that for very small logit values the output of the function is close to 0. While for large values of the logit, the output is close to 1. Between 0 and 1, the outputs are as shown in the Figure 2.4.

Another kind of neuron is the tanh neuron, which also uses a similar S-shaped curve but with values ranging from -1 to 1. This is unlike the sigmoid function where values are

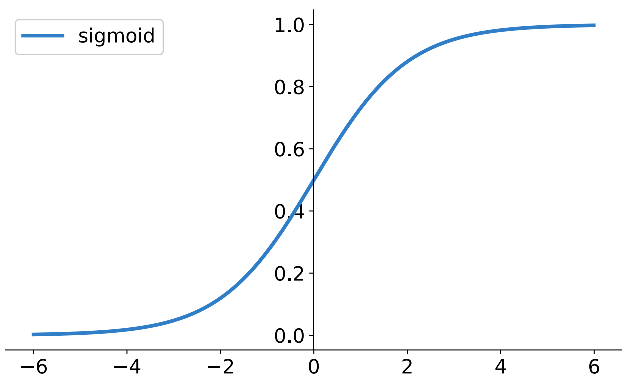


FIGURE 2.4: Output of sigmoid neuron as a function of z.

between 0 and 1. The function is, as expected, $f(z) = tanh(z)$. The relationship between the input logits and output values can be seen in Figure 2.5.

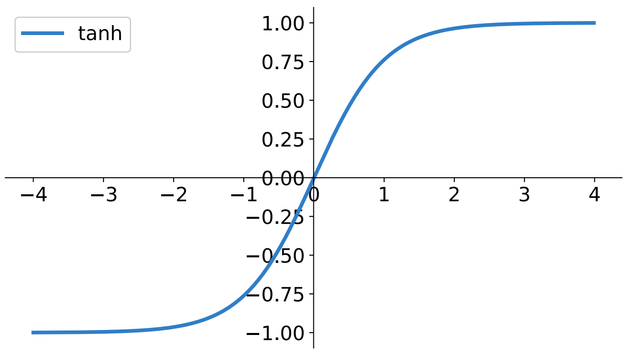


FIGURE 2.5: Output of tanh neuron as a function of z.

The last type of neuron we are going to discuss is the restricted linear unit (ReLU) neuron. This function is given by $f(z) = max(0, z)$ resulting in the graph shown in Figure 2.6.

Other activation functions also exist, for example, like exponential linear unit (ELU), binary step, SoftPlus, Leaky ReLU [40]. The ones described in this report are the most commonly used.

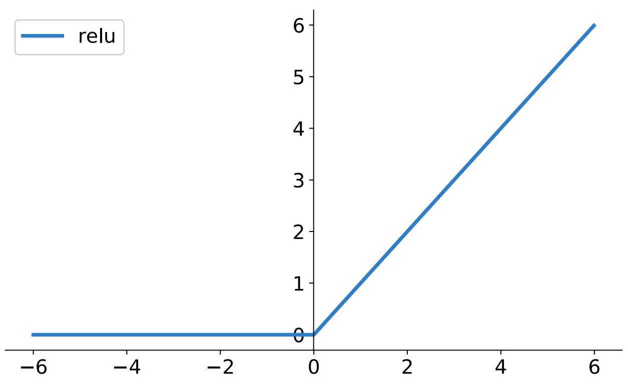


FIGURE 2.6: Output of ReLU neuron as a function of z.

2.2.3 Training

A deep learning neural network learns to map a set of inputs to outputs from training data. As there are a large number of unknowns, we cannot calculate the perfect set of weights for the network. Instead, training a neural network is treated as an optimization problem and an algorithm is used to find a set of weights which make relatively good predictions. Before an optimization algorithm is employed we need a metric to measure how well the weights are doing at every step and updating them accordingly. This metric is our loss function and the goal of the training process is to minimize this loss function. There is no one loss function which can be used for all machine learning problems. Choice of the loss function depends primarily on the type of algorithm we are using and the ease with which the function's derivative can be calculated [41].

There are two major types of loss functions: regression losses, and classification losses [42]. In regression, the model deals with predicting a continuous variable like stress, strain for a particular grain. Classification, on the other hand deals with predicting the output from a set of finite categorical value, example labeling a grain as a hotspot. A detailed description of loss functions are given below:

1. **Mean Square Error (MSE)**: is computed from the average of squared difference

between actual observations and predictions. The squaring enables heavy penalization of predictions that are far away from the true value. MSE has favorable mathematical properties of convexity, symmetry, and differentiability. Minimising MSE often has a closed-form analytical solution, and when they do not, iterative numerical optimization procedures are often easy to formulate, since the gradient is easy to compute [43]. If y_i is the true value and $\hat{y_i}$ is the predicted value then it can be calculated as:

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y_i})^2}{n} \tag{2.5}$$

2. **Mean Absolute Error (MAE)**: is computed as the averaged sum of absolute difference between actual observations and predictions. However MAE, has proved to under-fit training date due to it's low variance over data points [44]. It is given by:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - \hat{y_i}|}{n} \tag{2.6}$$

3. **Cross Entropy Loss**: is commonly used for classifications problems. Cross-entropy loss increases as the predicted probability diverges from the actual label. The mathematical formulation is given as:

$$CrossEntropyLoss = -(y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})) \tag{2.7}$$

Having described the loss functions, we now describe an algorithm to find the weights and biases so that the network's output approximates well for all training inputs. We start with looking at a simpler version of the mean squared error cost function and later generalize for a neural network. The cost function can be written as follows:

$$C(\boldsymbol{w}, \boldsymbol{b}) = \frac{\sum_{x} ||y(x) - a||^2}{n} \tag{2.8}$$

Here, $\boldsymbol{w}$ and $\boldsymbol{b}$ denote the weights and biases respectively, n is the number of training inputs and a is the output from the network. The aim of the optimization algorithm is to minimise this cost function: the training algorithm has done well if it can find weights and biases such that $C(\boldsymbol{w}, \boldsymbol{b}) \approx 0$. To achieve this we use a gradient descent algorithm. Currently the problem looks very complicated - the interpretation of $\boldsymbol{w}$ and $\boldsymbol{b}$ and the summation function in the background. Let us ignore most of these structures and just concentrate on the minimization aspect. Let us further assume that we have a function of two variables w_1 and w_2, and we want to minimise this function. So our new function is $C(w_1, w_2)$, and for small changes Δw_1, Δw_2 the corresponding change in $C(w_1, w_2)$ can be given as:

$$\Delta C \approx \frac{\partial C}{\partial w_1}\Delta w_1 + \frac{\partial C}{\partial w_2}\Delta w_2 \tag{2.9}$$

We can define vector as $\boldsymbol{\Delta} w = (\Delta w_1, \Delta w_2)^T$ and the gradient vector by $\boldsymbol{\nabla} C$. The gradient vector, $\boldsymbol{\nabla} C$, is the direction of steepest descent for any function, hence we move along this direction.

$$\boldsymbol{\nabla C} = \left(\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2}\right)^T \tag{2.10}$$

With these definitions, Equation 2.9 can be written as

$$\Delta C \approx \boldsymbol{\nabla C} \cdot \boldsymbol{\Delta} w \tag{2.11}$$

We want ΔC to always be negative because we want to minimise the function. We can choose our Δv such that ΔC is always negative. [45]

$$\boldsymbol{\Delta} w = -\eta \boldsymbol{\nabla} C \tag{2.12}$$

η is a small positive character known as the learning rate. It is a hyperparameter that controls how much one should adjust the model in response to the estimated error everytime the weights are updated. Very small a value may cause results in a long training process, whereas a large value may always yield a sub-optimal set of weights and never converge [46]. Selection of learning rate is also another crucial step in the training process. Moving on, from Equation 2.11 we can say $\Delta C \approx -\eta \boldsymbol{\nabla} C \cdot \boldsymbol{\nabla} C = -\eta ||\boldsymbol{\nabla} C||^2$. As $||\boldsymbol{\nabla} C||^2 \geq 0$, this guarantees that $\Delta C \leq 0$, i.e., C will always decrease. Thus, the weights can be updated using the following rule:

$$\boldsymbol{w} \rightarrow \boldsymbol{w}^{'} = \boldsymbol{w} - \eta \boldsymbol{\nabla} C \tag{2.13}$$

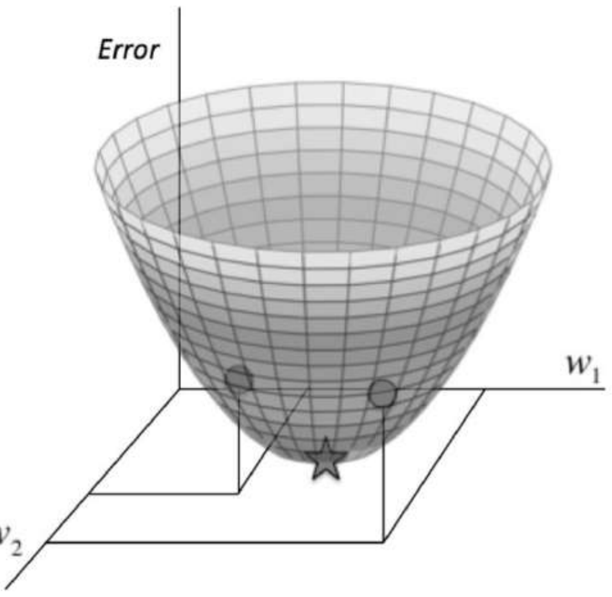


FIGURE 2.7: Error surface as a function of w_1 and w_1

The above example as we spoke earlier is a very simplified case with just two weights taken into consideration. But actual neural networks have a large number of weights and biases segregates into different layers. We need a systematic algorithm to update all the values according to the error. Back-propagation helps in understanding how changing the weights and biases changes the cost function. Before we get started let us define some notations: w^l_{jk} denotes the weight for the connection from k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. Similarly for biases, b^l_j for the bias of j^{th} neuron in the l^{th} layer. a^l_j will be used for the activation. L denotes the last layer in the network. The goal of back-propagation is to compute partial derivatives, $\partial C/\partial w$ and $\partial C/\partial b$, with respect to any weight or bias. Let us say we introduce an error Δz^l_j in the logit of the j^{th} neuron in the layer l. This will change the output give by this neuron's activation function and propagate through the later layers finally cost function to change by $\frac{\partial C}{\partial z^l_j}\Delta z^l_j$. Thus in some sense we can define the error in that neuron by

$$\delta^l_j = \frac{\partial C}{\partial z^l_j} \tag{2.14}$$

$\boldsymbol{\delta^l}$ will denote the vector of errors assiciated with layer l. Using backpropagation we will be able to calculate $\boldsymbol{\delta^l}$ for all the layers and relates those to our variables of interest through, $\partial C/\partial w^l_{jk}$ and $\partial C/\partial b^l_j$. The backpropagation algorithm is based on four fundamental equations which give us a way of computing both the gradient of the cost function and error $\boldsymbol{\delta^l}$ [45].

1. **Error in output layers**: is given below. The $\partial C/\partial a^L_j$ measures how fast the cost function changes with the activation of the j^{th} neuron in the output layer. The second term accounts for changes in the activation function σ with changes in z^L_j.

$$\delta^l_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j) \tag{2.15}$$

The equation can be re-written for all neurons using matrix-based form using Hadamard product, which is basically the element-wise product of two vectors.

$$\boldsymbol{\delta^L} = \nabla_a C \odot \sigma'(z^L) \tag{2.16}$$

2. **Error in terms of error in the next layer**: is given below where (w^{l+1}) is the weight matrix for the $(l+1)^{th}$ layer. If we know the error $\boldsymbol{\delta^{l+1}}$ for $(l+1)^{th}$ layer, then by mutiplying the transpose of the weight matrix we move the error backward through the network to give us the measure of error at the l^{th} layer. By taking the Hadamard product with $\sigma'(z^L)$ the error moves backward through the activation layer, giving us $\boldsymbol{\delta^l}$ for that layer.

$$\boldsymbol{\delta^l} = ((w^{l+1})^T\delta^{l+1}) \odot \sigma'(z^l) \tag{2.17}$$

Using the equations above, we can calcualte the $\boldsymbol{\delta^l}$ for any layer starting with the output layer.

3. **Change of error with bias**: is a simple equation given by:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.18}$$

4. **Change of cost with any weight**: can now easily be given by the following equation. We already know how to compute $\boldsymbol{a^{l-1}}$ and $\boldsymbol{\delta^l}$.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{2.19}$$

2.2.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) have played an important role in the field of machine learning ever since their introduction and development in the 1990s. They are used to learn time-varying and sequential patterns. [47] RNNs leverage a special type of neural layer, known as recurrent layers which makes different from a neural network. The recurrent layer enables the network to maintain state between the uses of the network. All the neurons in an RNN have three types of connections: (i) incoming connections from the neuron of the previous layer (input) (ii) outgoing connections leading to neurons in the next layer (output), and (iii) recurrent connections between neurons in the same layer. Thus, a fully connected recurrent layer has information flowing from every neuron to every other neuron in the same as well as the next layer (including itself). A recurrent layer with r neurons has r^2 connections within the same layer.

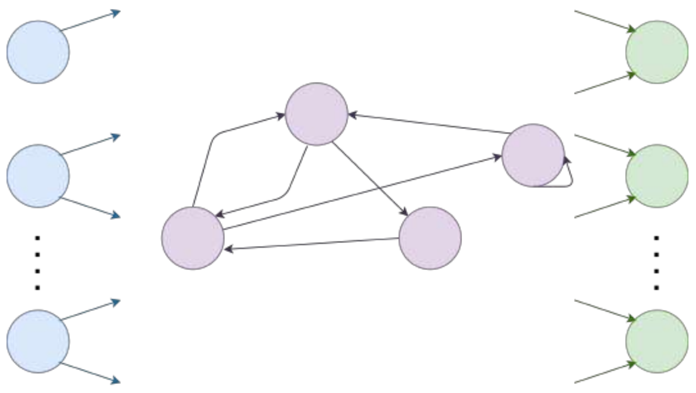


FIGURE 2.8: A recurrent layer with connections between neurons in the same layer.

RNNs are a category of neural networks where output from the previous time steps is taken as input for the current time-step. A RNN instance can actually be expressed as a feed forward neural network for a given fixed lifetime (t time-steps). This transformation is referred to as "unrolling" the RNN through time. The update graph formed by unrolling is a useful way to visualise RNNs. The transformation is performed by repeating the neurons in the recurrent layer t times, once for each time-step. Similarily the input and output connections are redrawn as they were in the original network and all the feed forward connections are made. Finally, feed forwards connections are drawn from each time-step replica to the next time-step as recurrent neural networks use neuronal activation from the previous time-step. Consider an input with T timesteps, I input units, H hidden units, and K output units. Let x_i^t be the input i at time t, z_j^t be the input logit at of unit j and y_j^t be the activation. Then for hidden units, the logit can be written as

$$z_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} y_{h'}^{t-1} \tag{2.20}$$

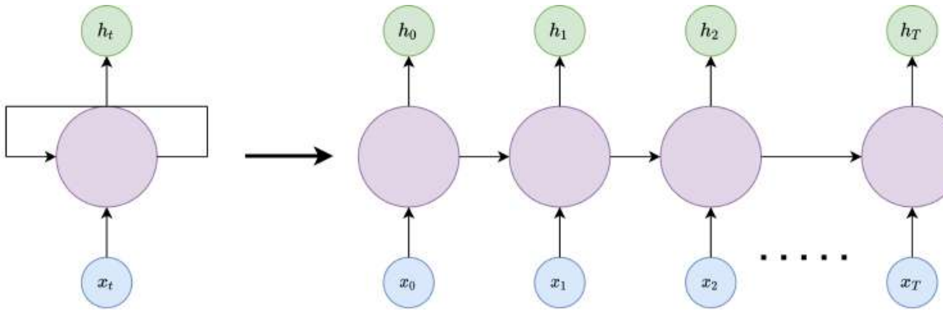


FIGURE 2.9: RNN unrolled over time

The unrolled RNN can be trained by computing the gradient and using the techniques used for a feed forward neural network. However, for standards RNN architectures the range of context that can be accessed is quite limited. The influence of the inputs on the hidden layers either decays or blows exponentially as it cycles around the network's recurrent connections. This is often referred to as the problem of vanishing gradient [48]. It is depicted in the figure given below. The shade of the node in the unrolled RNN indicates the relevance of the input at the first time step (darker the shade, greater the sensitivity). Numerous attempts were made to solve this problem, one of which is long short-term memory networks.

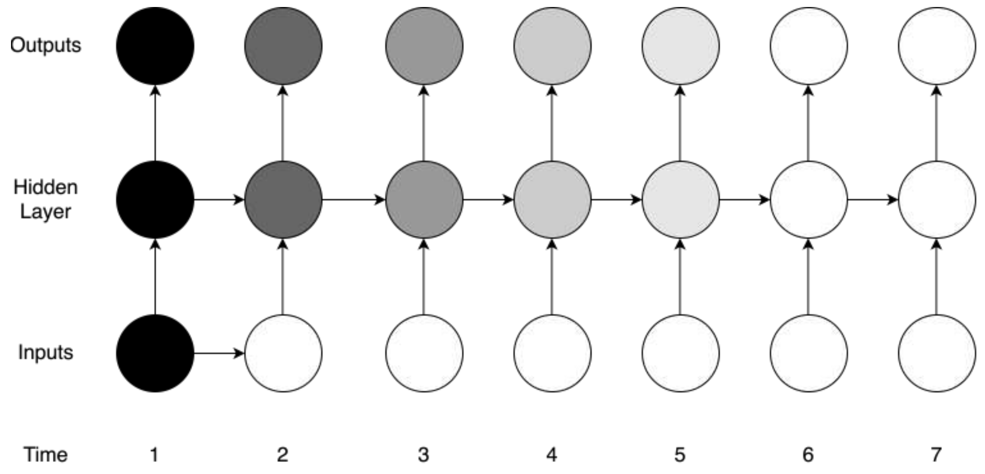


FIGURE 2.10: Vanishing gradient.

2.2.5 Long Short-Term Memory (LSTM) Networks

Long short-term memory networks were introduced by Hochreiter and Schmidhuber with the basic principle that the network would be designed for transmitting useful information multiple time-steps into the future [37]. The LSTM architecture consists of memory cells, which are recurrently connected sub-nets. This memory cell is responsible for holding important information that the network has learnt over time and the network is designed to maintain this information with time. The memory cell functions with the help of three gates, which are discussed next. The schematic of a LSTM unit $\hat{y}_i$ can be seen in figure-2.11.

The memory cells have a keep gate, shown in figure-2.12, which determines how much of the previous memory is useful and will be stored for future use. Memory state from the previous time-step is stored in the form of a tensor, rich in information. To figure out the elements in the memory tensor which are still useful, a bit tensor (tensor of zeros and ones) is calculated that we multiply to the memory state tensor. Intuitively, a zero means

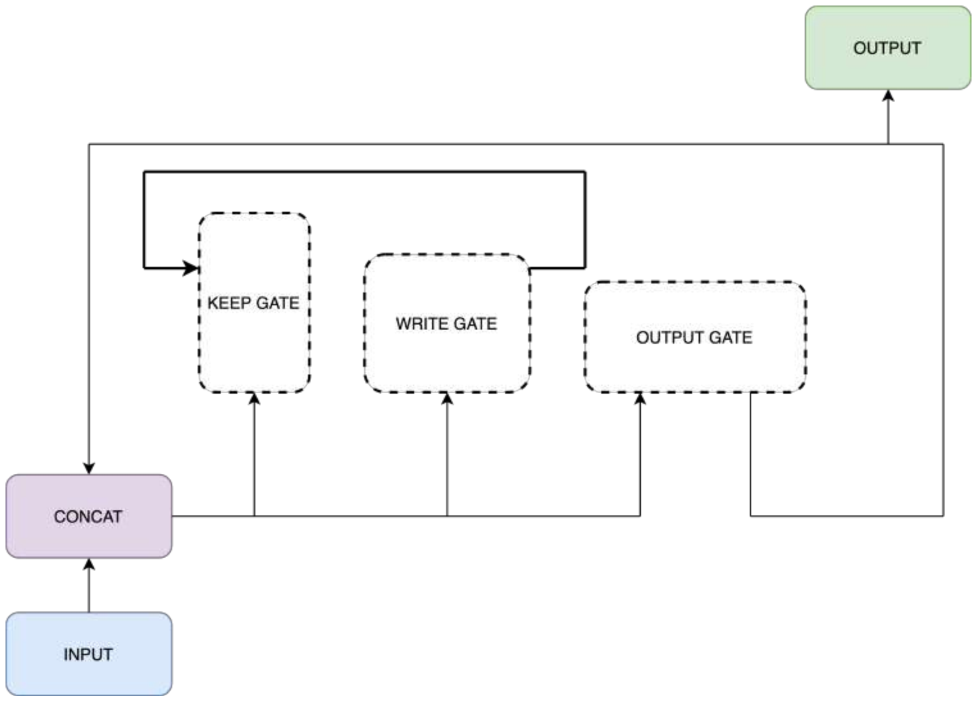


FIGURE 2.11: A LSTM unit.

that the element's information is useless and one means that it is useful. The bit tensor is approximated by concatenating the output from the previous time-step along with the input of this time-step and applying a sigmoidal activation to them. The sigmoid function outputs value very close to 1 or 0 and hence works well for this task.

Once the LSTM knows which information is useful, it is ready to write into the memory state, shown in figure-2.13. This part is handled by another LSTM unit called the write gate. The write gate's function can be broken down into two major parts. Figuring out what information to write into the state is the first step. This is computed by concatenating the output from the previous time-step along with the input of this time-step and applying a tanh activation to them. The second part involves figuring part which parts are relevant and need to be stored in the new state. A similar strategy is used as the keep gate by creating a bit vector and multiplying it with the intermediate tensor. The result is added with the output of the keep gate to create a new state.

At the final step, the LSTM unit provide it's output. The structure of the output gate, shown in figure-2.14, is similar to the write gate. Tanh activation is applied to the state

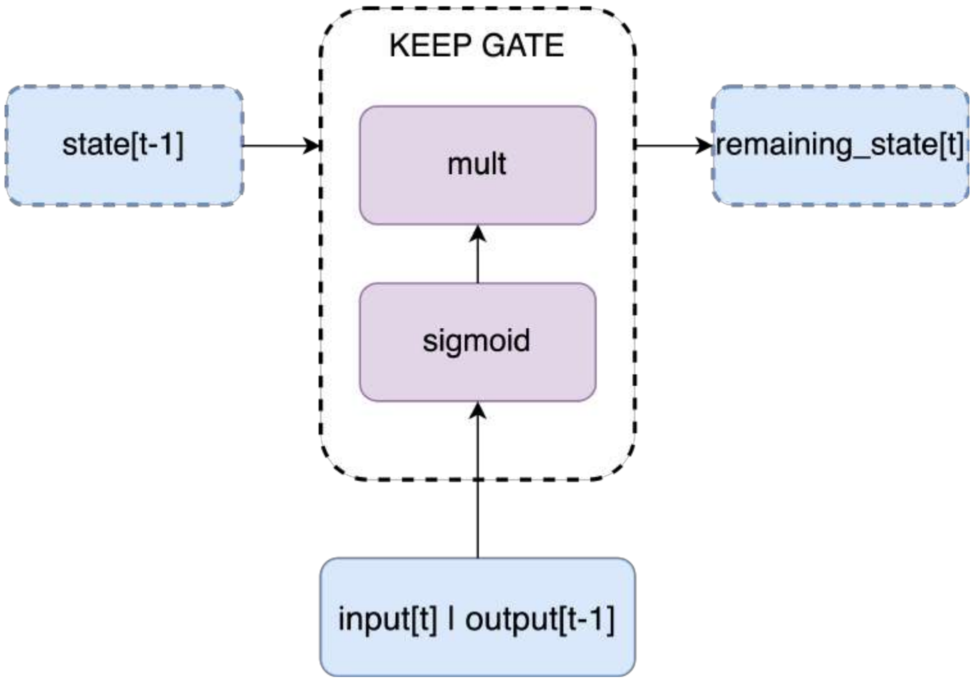


FIGURE 2.12: Architecture of the keep gate of the LSTM unit.

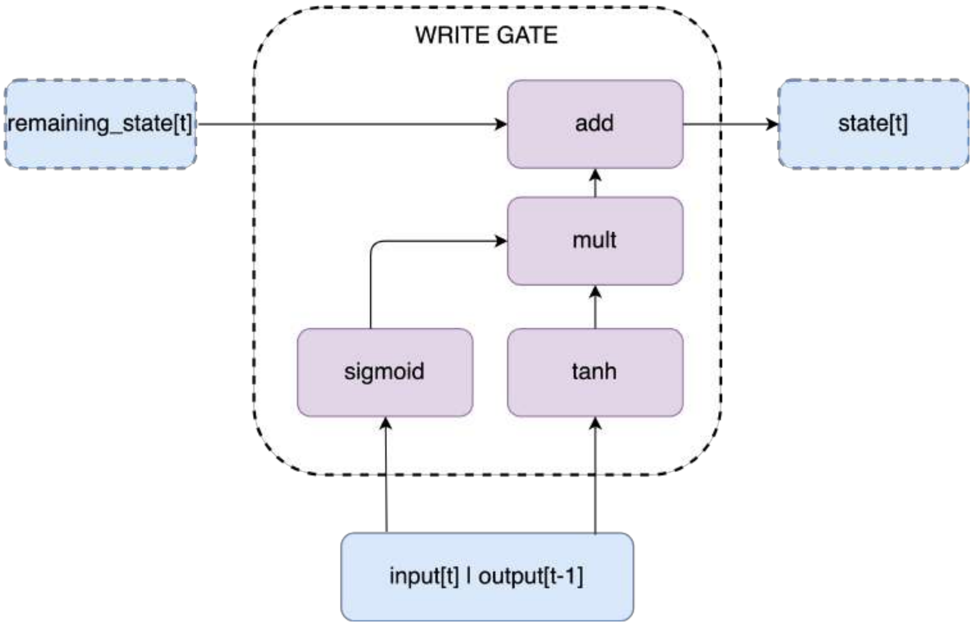


FIGURE 2.13: Architecture of the write gate of the LSTM unit

vector to form an intermediate tensor. A sigmoidal layer is used to create the mask using the current input and previous output. The intermediate tensor is multiplied by the bit tensor(mask) to produce the final output.

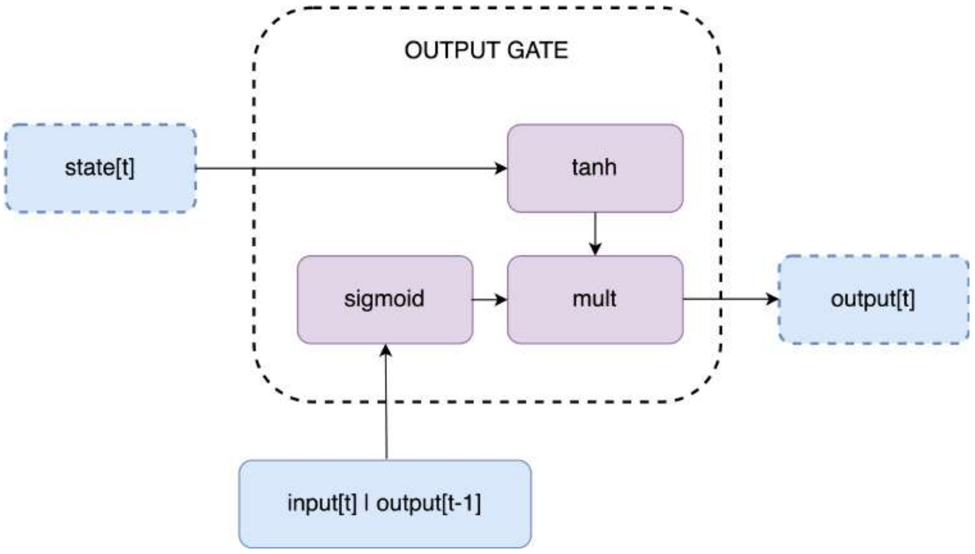


FIGURE 2.14: Architecture of the output gate of the LSTM unit.

The key equations involved in LSTM are given below where $\odot$ represents the Hadamard product, tanh is applied element-wise, x_t, c_t and h_t are the input, memory cell status and output of the LSTM at time t. i_t, o_t and k_t are the functional values of the write, output and keep gate. Ws denote the weight matrix between the input and the gate and bs are the biases.

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci} \odot c_{t-1} + b_i\right) \tag{2.21}$$

$$k_t = \sigma\left(W_{xk}x_t + W_{hk}h_{t-1} + W_{ck} \odot c_{t-1} + b_k\right) \tag{2.22}$$

$$c_t = f_t \odot c_{t-1} + tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \tag{2.23}$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co} \odot c_{t-1} + b_o\right) \tag{2.24}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.25}$$

Chapter 3

Model Development

3.1 Finite Element Modeling Framework

3.1.1 Microstructure Instantiation

We have used an in-house MATLAB code to instantiate the 2D microstructures of a dual-phase (DP) steel comprising of ferrite and martensite phases. This code employs a weighted Voronoi tessellation algorithm for random positioning of grain centers within the simulation domain. The grain size distribution and volume fractions of the respective phrases are pre-defined based on experimental measurements. Figure-3.1 shows the grain map of the microstructure generated by this algorithm. The desired mean radius r_i and volume fractions V_i are given as inputs to the algorithm. The number of grains are calculated by the algorithm as $(Area \times V_i)/(\pi r_i^2)$. The sizes of these n_i grains are generated randomly. The subscript i has a value of 1 for ferrite and 2 for martensite. Details of the microstructure instantiation algorithm are given in Ref. [49]. The Trelis meshing software has been used to mesh the 2D microstructure with voxel-based elements. An element size of $0.25\mu m$ is used to mesh the simulation domain of $100 \times 100\mu m^2$.

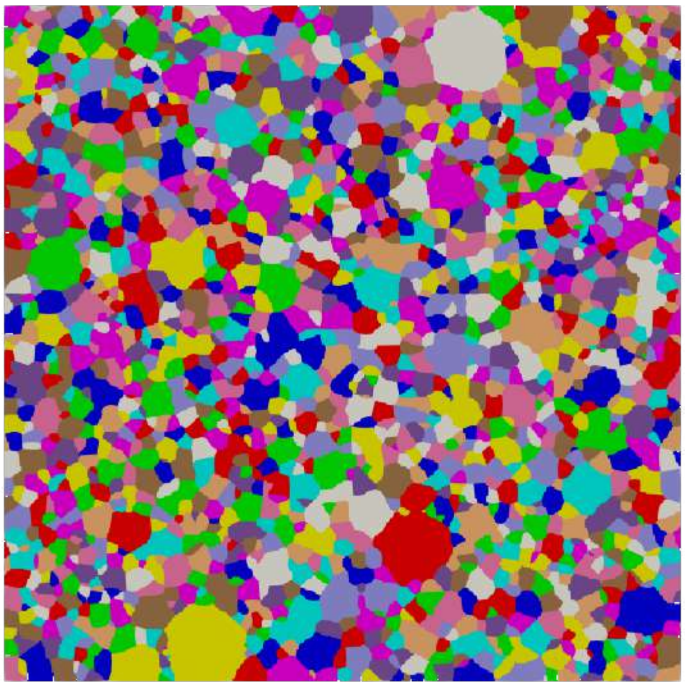


FIGURE 3.1: Grain Map (colours are random and only to demarcate one grain from another)

3.1.2 Constitutive Model

We have used a J2 plasticity constitutive modeling framework for the finite element (FE) computations. This framework is based on a finite deformation, dislocation density based model adapted from a crystal plasticity model previously used to model deformation behavior of various metallic systems [20] [6]. While this is not a new feature of this work, details are provided here for completeness.

In this framework, the deformation gradient is multiplicatively decomposed into the elastic and plastic parts:

$$\boldsymbol{F} = \boldsymbol{F^e} \cdot \boldsymbol{F^p} \tag{3.1}$$

where, $\boldsymbol{F^p}$ relates the reference configuration to an intermediate configuration and accounts for shear due to plastic deformation, while $\boldsymbol{F^e}$ relates the intermediate configuration to the current, deformed deformation and accounts for the rigid body rotation and elastic distortion. The elastic Green strain is given as:

$$\boldsymbol{E^e} = \frac{1}{2}(\boldsymbol{F^{eT}} \cdot \boldsymbol{F^e} - \boldsymbol{I}) \tag{3.2}$$

Further, the second Piola-Kirchoff tensor is related to the elastic Green strain via the fourth rank elasticity tensor $\boldsymbol{C_o}$, i.e.,

$$\boldsymbol{\sigma^{PK2}} = \boldsymbol{C_o} : \boldsymbol{E^e} \tag{3.3}$$

The second Piola-Kirchoff tensor $\boldsymbol{\sigma^{PK2}}$ and the cauchy stress $\boldsymbol{\sigma}$

$$\boldsymbol{\sigma} = \frac{\boldsymbol{F^e} \cdot \boldsymbol{\sigma^{PK2}} \cdot (\boldsymbol{F^e})^T}{det\,(\boldsymbol{F^e})} \tag{3.4}$$

The von Mises effective stress, $\bar{\sigma}$ can be given in terms of the deviatoric part $\boldsymbol{S}$ of the Cauchy stress, i.e.,

$$\boldsymbol{S} = dev((\sigma)) = \boldsymbol{\sigma} - \frac{(tr(\boldsymbol{\sigma}))\,\boldsymbol{I}}{3} \tag{3.5}$$

$$\bar{\sigma} = \sqrt{\frac{3}{2}\boldsymbol{S} : \boldsymbol{S}} \tag{3.6}$$

In J2 plasticity, the equivalent plastic strain rate $\dot{\bar{\epsilon}}^p$ is given as a function of the von Mises effective stress $\bar{\sigma}$. In this finite deformation framework, the plastic deformation gradient is related to the plastic velocity gradient as: $\dot{\boldsymbol{F}}^{\boldsymbol{p}} = \boldsymbol{L^p} \cdot \boldsymbol{F^p}$, where $\boldsymbol{L^p}$ is the velocity gradient given by:

$$\boldsymbol{L^p} = \sqrt{\frac{3}{2}}\dot{\bar{\epsilon}}^p \boldsymbol{N^p} \tag{3.7}$$

where $\dot{\bar{\epsilon}}^p$ is the effective plastic strain, and $\boldsymbol{N^p}$ gives the direction of the plastic flow given by:

$$\boldsymbol{N^p} = \sqrt{\frac{3}{2}}\frac{\boldsymbol{S}}{\bar{\sigma}} \tag{3.8}$$

The equivalent plastic strain rate is modelled using a Kocks-type activation enthalpy-driven flow rule:

$$\dot{\bar{\epsilon}}^p = \dot{\bar{\epsilon}}^p_o \exp\left(\frac{-\Delta F_g}{kT}\left(1 - \left(\frac{\sigma_{eff} - S_a}{S_t}\right)^p\right)^q\right) \tag{3.9}$$

where, ΔF_g is the activation energy for dislocation glide, S_a is athermal slip resistance, and S_t is the thermal slip resistance, p and q are parameters used to model the shape of the activation enthalpy curve. The athermal slip resistance S_a can be given by:

$$S_a = \frac{h_p}{\sqrt{d}} + Gb\sqrt{q_p \rho} \tag{3.10}$$

where, h_p is the Hall-Petch hardening constant, d is the grain size, G is the shear modulus, b is the Burgers vector, q_p is the dislocation barrier strength and ρ is the total dislocation density. ρ is defined as the sum of immobile (ρ_M) and mobile (ρ_I) dislocation densities, i.e.,

$$\rho = \rho_M + \rho_I \tag{3.11}$$

The evolution of the dislocation densities with respect to time depends on the equivalent tensile plastic strain rate as:

$$\dot{\rho_M} = \frac{k_m ul}{b}\sqrt{\Sigma\rho}|\dot{\bar{\epsilon}^p}| - \frac{2R_c}{b}\rho_M|\dot{\bar{\epsilon}^p}| - \frac{1}{b\lambda}|\dot{\bar{\epsilon}^p}| \tag{3.12}$$

$$\dot{\rho_I} = \frac{1}{b\lambda}|\dot{\bar{\epsilon}^p}| - k_{dyn}\rho_I|\dot{\bar{\epsilon}^p}| \tag{3.13}$$

where $\lambda = \frac{1}{\beta\sqrt{\rho}}$ is effective mean free path, β is a constant associated with dislocation trapping, k_{mul} is the dislocation multiplication rate constant and R_c is the critical capture radius. The first term in Eq. 3.12 represents the multiplication of mobile dislocations at existing dislocations, while the second term represents the mutual annihilation of dislocation dipoles, and third term is the trapping of mobile dislocations at barriers. k_{dyn} is the material constant associated with dynamic recovery of immobile dislocations due to thermally activated processes. The first term in Eq. 3.13 represents the rate at which mobile dislocations are trapped in barriers and become immobile, while the second term is the rate at which the immobile dislocations are annihilated due to dynamic recovery.

The constitutive model has been implemented as material model and interfaced with the open source finite element code, MOOSE [50]. The individual phases have been calibrated to the response of the ferrite and martensite phases based on available data in the literature. Further details and model parameters are given in Basu et al [51].

3.1.3 Loading And Boundary Conditions

A mesh size of $0.25\mu m$ is used with a simulation domain size of $100 \times 100\mu m^2$. The simulation domain has 40,000 elements.A generalized plane strain assumption is used in these essentially 2D simulations. The bottom face of the simulation domain is constrained in the y-direction, while the left face is constrained in the x-direction to resemble an axisymmetric model. The corner node common to both these faces is fully constrained to prevent rigid body motion. Displacement-controlled tensile loading is applied on the top face at a strain rate of $5 \times 10^{-3}s^{-1}$.

3.2 Data Extraction from FE Results

The simulation results of the FE model are output to an EXODUS data file. The EXODUS format is a binary file which is used for FE pre- and post-processing. Data used to define the finite element mesh, along with both the boundary conditions and load application points are includes in the file. The EXODUS format is advantageous as it combines the mesh data as well as the results data in a single file. This assures the user that the results are consistent with the model [52]. However, to access this data in a usable and easily interpretable way, special software are used. For the purpose of this project, we have used the Sandia Engineering Analysis Code Access System (SEACAS) developed by Sjaardema [53]. The library consists of packages which can convert an EXODUS file's data to different formats like text file and MATLAB data files. The Python package has been used for this project which makes reading the data from an EXODUS file viable through a python script. The package provides a number of predefined functions to extract different information from the EXODUS file. Using these functions, we have extracted all the data from the EXODUS file to a csv format. The csv format has all the element variable values

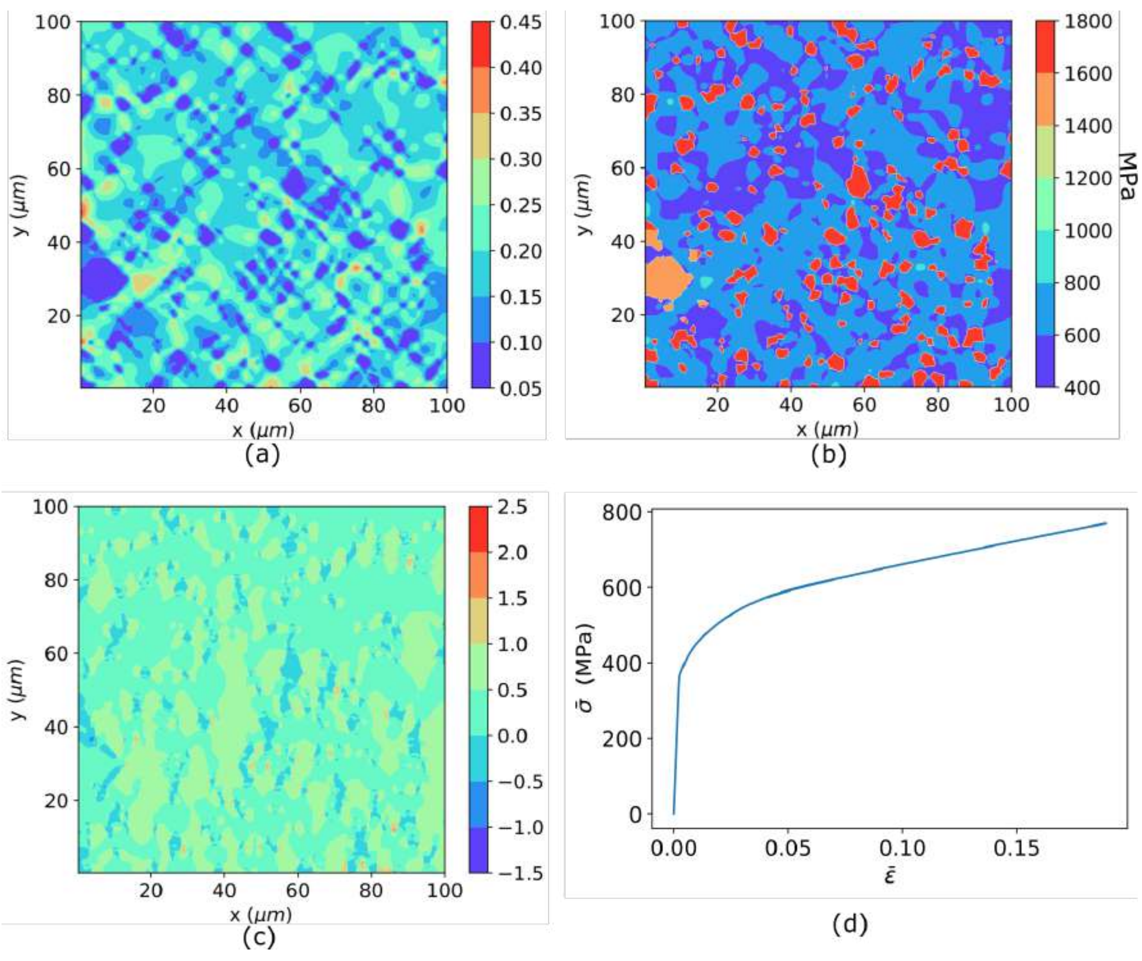


Figure 3.2: Contour of (a) Effective Strain (b) Triaxiality (c) Vonmises Stress at the last time step 9d) stress strain curve

at every time-step for every element. The script is written in Python 2.7.17 and take approximately 10 minutes to execute. The time taken by the script varies with the mesh size and number of variables in the exodus file.

3.3 Machine Learning Framework

3.3.1 ANN Model

At the basic level, the process of developing an ANN model (or any learning model) has two primary steps. First one being, selecting the relevant, representative and compact set of

Feature	Description
x	x-coordinate of the element
y	y-coordinate of the element
p	Phase to which the element belongs (ferrite or martensite)
ϵ_{app}	Applied (or nominal) strain

Table 3.1: Selected feature names and descriptions.

features and then developing a relation between them to get the desired result. Elimination of irrelevant features and selection or relevant ones is one of the central tasks in machine learning [54]. In this context, it has been assumed that the observed heterogeneity in deformation is due to the underlying heterogeneity of the microstructure. The features of interest are highlighted in the Table 3.1.

The output variables of interest are effective strain, $\bar{\epsilon}$, von Mises effective stress, $\bar{\sigma}$, and stress triaxiality ratio. While, effective strain and effective stress can be correlated with plastic deformation in the respective phases, the stress triaxiality ratio (given as the ratio of the mean stress over the von Mises effective stress) provides a measure of the development of multi-axial stress states that may contribute to eventual failure initiation in the material. In the present context, the ferrite-martensite interfaces are potential sites for failure initiation in DP steels and hence it is pertinent to be able to predict the stress triaxiality ratio as well. For each element, $\bar{\epsilon}$, $\bar{\sigma}$ and triaxiality ratio are recorded at applied strain intervals of 0.01. The simulation is run up to 0.15 nominal strain, thus providing us a total of fifteen datasets at 0.01 nominal strain intervals. The mesh consists of 400×400 elements, making the length of every feature and target variable vector $2,400,000$. Thus, the shape of our input is $2,400,000 \times 4$ and the corresponding shape of output is $2,400,000 \times 3$. The architecture of our ANN can be seen in 3.3

Note that feature selection depends on the output variables of interest and may vary if it is desired to output a different set of variables. Mangal and Holm [27] provide a detailed study of the performance of neural network models with different input features for a crystal plasticity model. In the context of a J2 plasticity model, we assume the features given in Table 3.1 to be the most relevant for the present problem.

Any ANN model is very sensitive to the magnitude of the training data. Given that we are interested in strains and stresses, whose magnitudes are generally many orders apart, all the data has been normalized within the range $[0, 1]$ in the present work. The normalised values x_i^n for any training variable x_i are calculated as:

$$x_i^n = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{3.14}$$

In this work, the mean-squared error function (MSE) has been used and the sigmoid function has been employed as the activation function for every neuron. The completed training and model development process has been performed using python and Keras with a Tensorflow backend. Data splitting is another important task during data splitting where hold-out validation is employed to ensure generalisation [55]. The dataset is split into training (80%) and testing (20%) datasets. The training data is used to for training and learning of the ANN, whereas the testing dataset is used to check the accuracy of the fitted parameters. The validation dataset is unseen by the ANN model, therefore predictions made by the model on the validation dataset represents the degree to which the model generalizes over our data. A number of networks were trained on this data with different architectures. The network which performed relatively better than others consisted of 4 inputs, 3 hidden layers and 3 outputs. The training of the neural network is performed over 5 epochs and takes 106 minutes to train. The evolution of loss with training can be seen in 3.4

3.3.2 LSTM Models

LSTMs have shown great accuracy in predicting sequence and time series data. The evolution of local strain distribution can be formulated as a sequence modeling problem. As with ANN, we are focusing on three output variables in our model, namely, effective strain, $\bar{\epsilon}$, von Mises effective stress, $\bar{\sigma}$, and stress triaxiality ratio. For every element, the values of these variables are recorded for every 0.01 nominal strain interval. As mentioned previously, the simulation is run till 0.15 nominal strain and hence our data contains 15

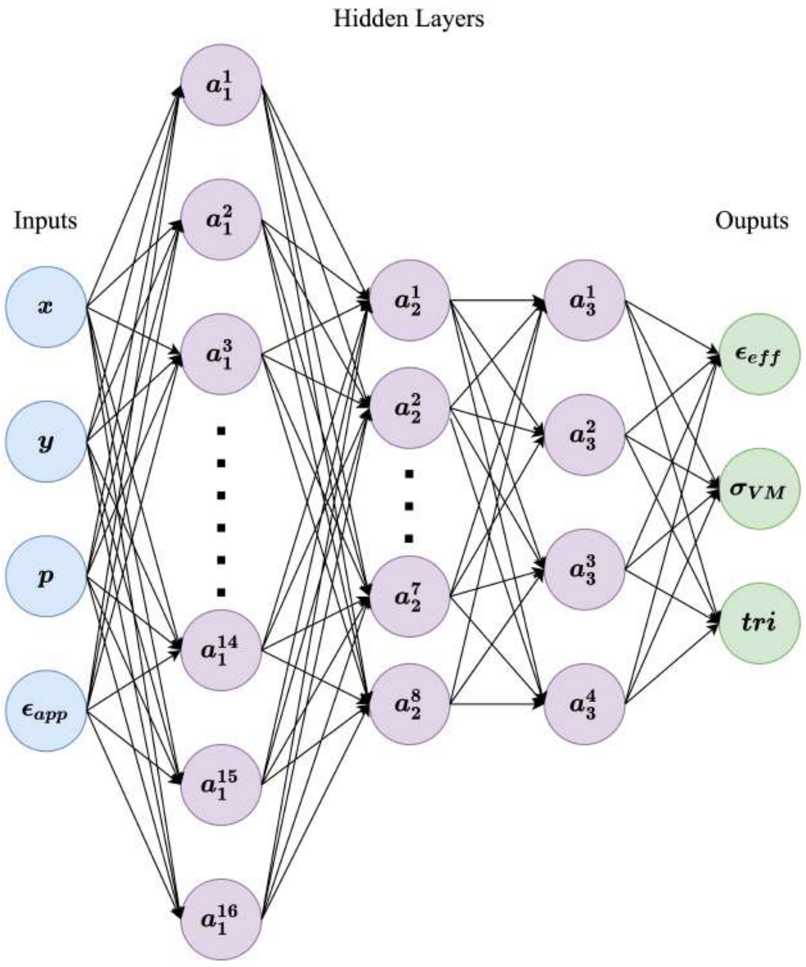


FIGURE 3.3: Schematic of proposed ANN architecture

strain steps. Multiple models and architectures were tried out with an LSTM, the details for which are as follows.

Each element is associated with 3 sequences of length 15, and as there are 400×400 elements the total number of series or model predicts is $400 \times 400 \times 3$. For each element, the first strain step values were used an input to predict 13 strain steps into the future. The intermediate values between the 1st and 15th strain step were also predicted and accounted for. All data was normalized within the range of $[0, 1]$. The data was split into training (70%), test (20%) and validation (10%) datasets. This model is trained such that given the data for a microstructure at 1% strain, it can predict values for effective strain, $\bar{\epsilon}$, von Mises effective stress, $\bar{\sigma}$, and stress triaxiality ratio upto 15% strain. The

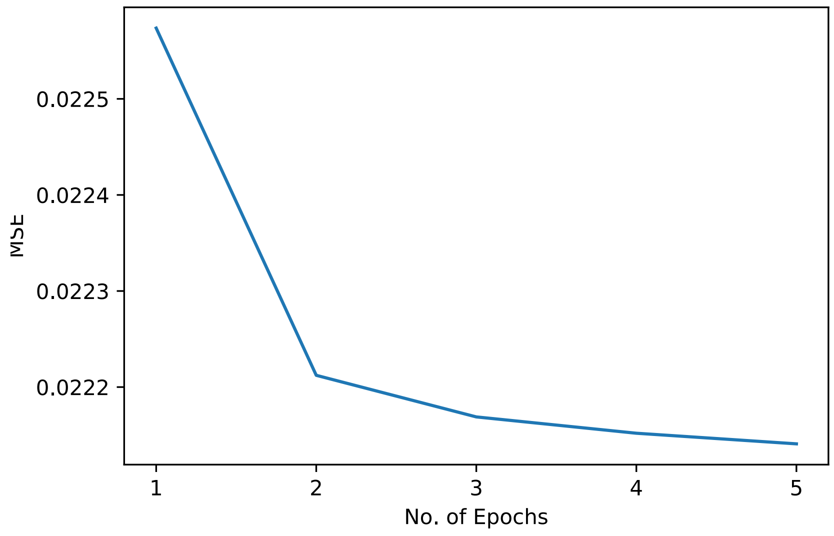


FIGURE 3.4: Learning curve showing the evolution of MSE loss over training data-set

model consists of 8 LSTM layers with 100 neurons in each layer. The activation function employed in this case is ReLU. The choice of activation function is driven by the reduction in MSE over the epochs. Figure 3.6 shows the performance of different activation functions while training the same dataset. The number of layers and neuron are the same for these model, only the variation with activation function is checked for. It can be observed that ReLU performs better than the other two activation functions by at least two orders of magnitude of MSE. The evolution of loss with training can be seen in 3.5. The total number of trainable parameters in the LSTM were 604,703.

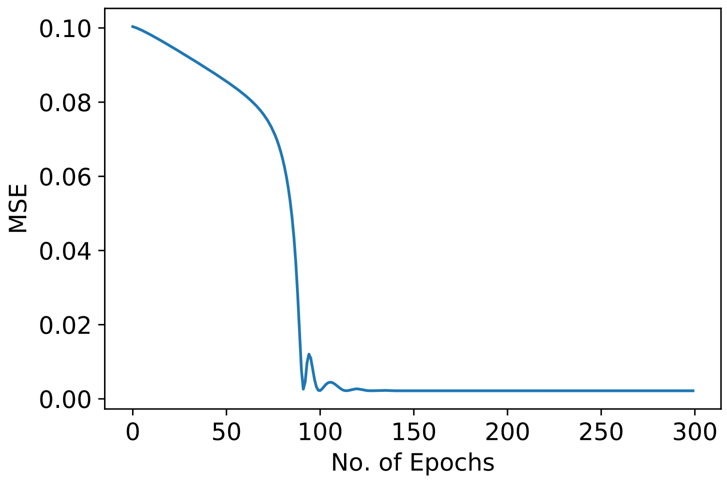


FIGURE 3.5: Learning curve showing the evolution of MSE loss over training data-set.

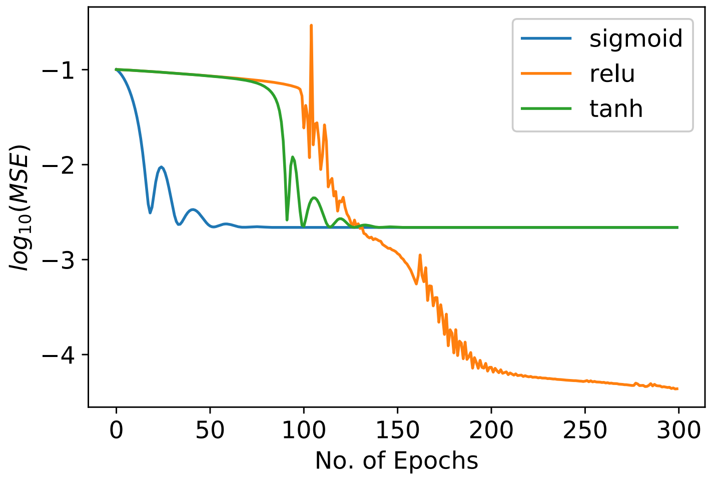
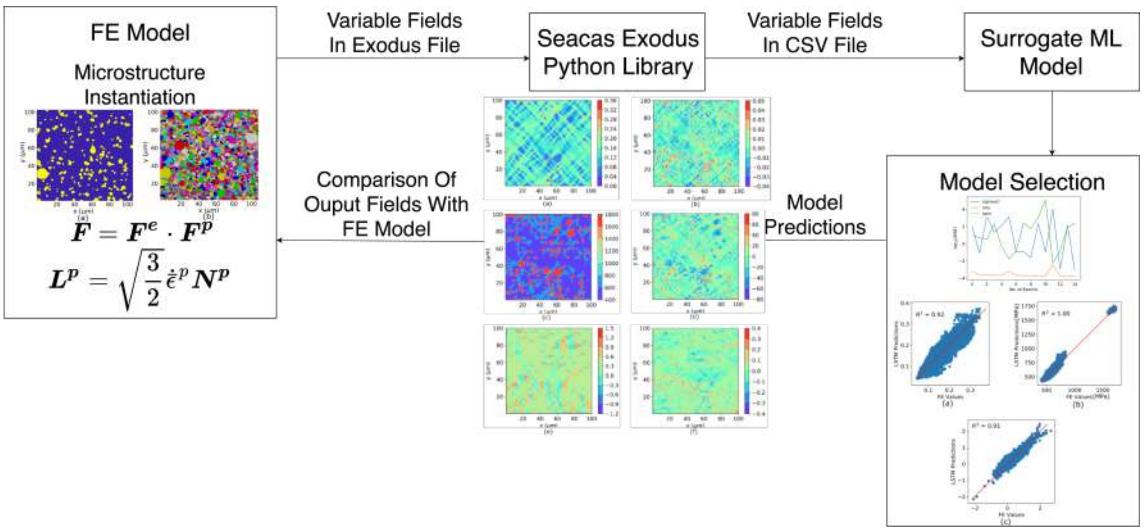


FIGURE 3.6: Comparative study of MSE obtained in training models with different activation functions



FE Model
Microstructure Instantiation
$F = F^e \cdot F^p$
$L^p = \sqrt{\frac{3}{2}}\dot{\bar{\epsilon}}^p N^p$
Variable Fields In Exodus File
Seacas Exodus Python Library
Variable Fields In CSV File
Surrogate ML Model
Model Selection
Model Predictions
Comparison Of Ouput Fields With FE Model

FIGURE 3.7: Comparative study of MSE obtained in training models with different activation functions

Chapter 4

Results and Discussion

4.1 Artificial Neural Network (ANN) Results

The relationship between the microstructure and mechanical behaviour of DP steels is extremely complex. Successful implementation of an ANN model would have helped understand these complex relations and helped in predicting experimental behavior of other heterogeneous systems. For assessing our model's predictive capabilities over three different variables simultaneously, a metric is used to estimate how well the model performs for each of these variables. For this purpose, we calculate the coefficient of determination, R^2. R^2 represents the proportion of variance that has been explained by the independent variables in the model. It provides an indication of goodness of fit and therefore a measure of how well unseen samples are likely to be predicted by the model, through the proportion of explained variance. Best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). This metric is given by the following equation:

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.1}$$

where y is the actual output (true value) and $\hat{y}$ is the prediction made by the ANN model. For a model with 100% accuracy then $y_i = \hat{y}_i)$ for all $i's$ making $R^2 = 1$, which indeed is

Layers	Architecture	MSE	Time	R^2		
	(Neurons in every layer)		(hrs)	$\bar{\epsilon}$	$\bar{\sigma}$	tri
2	8,4	0.0221	1.44	0.55	0.16	0.7
3	16,8,4,	0.0221	1.76	0.55	0.16	0.07
4	32,16,8,4	0.0221	1.78	0.55	0.17	0.06
5	64,32,16,8,4	0.0222	1.82	0.55	0.15	0.05
6	128, 64,32,16,8,4	0.0266	1.87	0.16	0.13	0.02
7	256,128,64,32,16,8,4	0.0290	3.47	< 0	< 0	< 0
8	512,256,128,64,32,16,8,4	0.0290	4.59	< 0	< 0	< 0
9	1024,512,256,128,64,32,16,8,4	0.0289	7.32	< 0	< 0	< 0
11	1024,1024,512,512,256,128,64,32,16,8,4	0.0290	20.24	< 0	< 0	< 0

Table 4.1: Study of accuracy of models with different depths and neurons.

the best case scenario.

A number of neural network architectures were trained and their R^2 and mean squared error values were recorded for a comparative parametric study. The following table shows the results that were obtained along with the time taken to train each model.

As can be clearly seen from the table, the model is unable to make the predictions successfully. The MSE is very high and the R^2 values too low. For the model to be acceptable, the MSE values should have been in the range of $10^{-4} - 10^{-5}$ and R^2 values should be above 0.80. All the above simulations have run for 5 epochs. Increasing the epochs is usually accompanied by a decrease in error. However, even after increasing to 20 epochs, the R^2 values did not improve. The first architecture in Table 4.1 was trained for 20 epochs. The MSE reduced to 0.0216, but R^2 values remained approximately the same. Increasing the epochs increased the training time to 8.5 hours without yielding any significant results. Increasing the dept and complexity of the network is another way to increase the accuracy. However, as can be seen from the table, the MSE and R^2 values actually worsen as the number of hidden layers are increased beyond 4. Another major drawback in the ANN model is the limited dataset we are using to train it. The data consists of only discrete data at 0.01 strain intervals. If instead, we trained our data to smaller strain intervals of the FE simulations, the results would be perhaps better, albeit at the cost of significant training times.

ANN predicted values are plotted in Figure 4.1 against the ground truth values to better visualize the performance of our model. Note that all results are plotted on the normalized scales. For an ideal case, all the data points would lie as close to the $y = x$ line. A separate

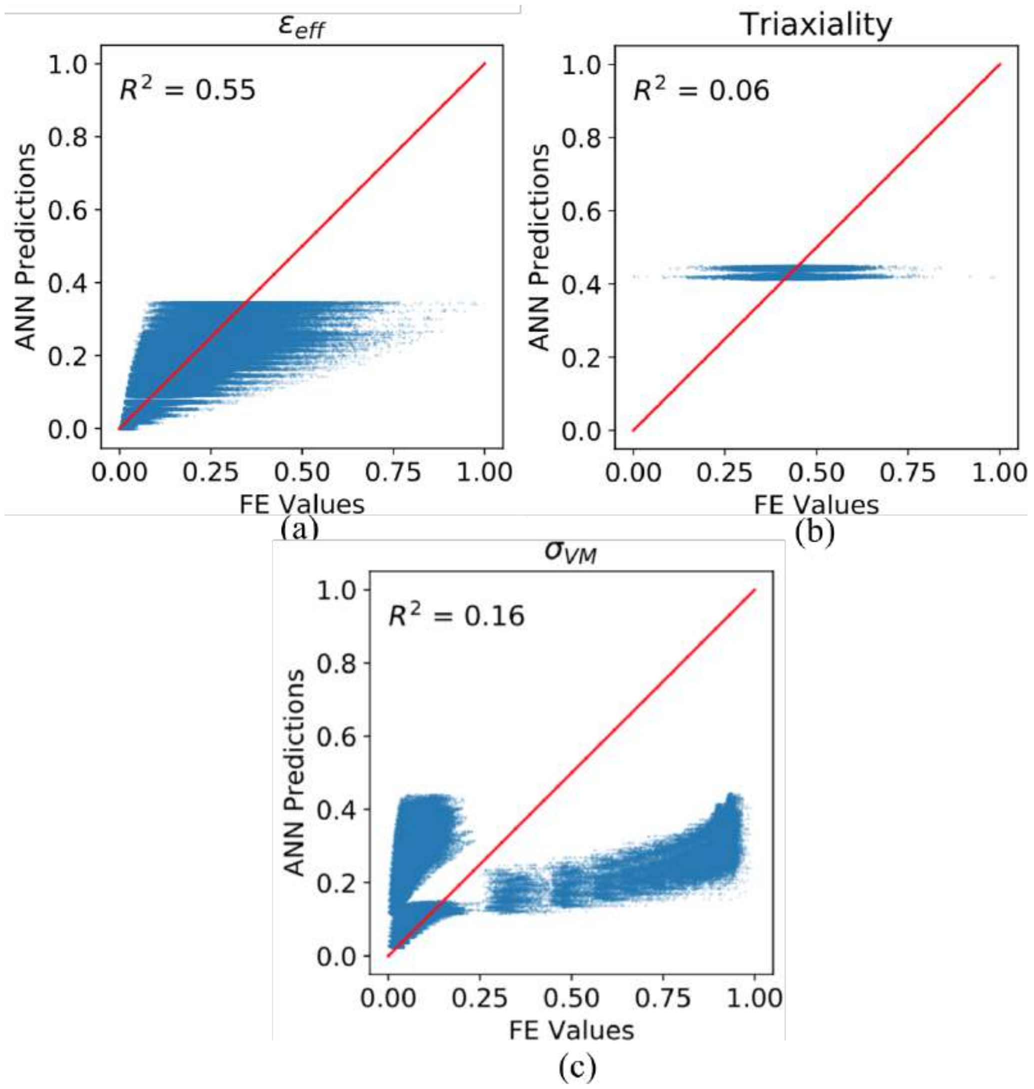


Figure 4.1: Plot of ANN predicted values vs FE values for (a) effective strain, (b) stress triaxiality ratio, and (c) von Mises effective stress. Note that all the variables are normalized.

model was also trained to predict only one variable, effective strain. The architecture was kept the same, and the results showed an improvement in the R^2 value. We can conclude that predicting multiple variables using a single ANN leads to inaccuracy. Figure-4.2 shows the R^2 values and figure-4.3 shows the contour plots of ANN predicted value as compared to the true values. The numerical value of the predictions is quite inaccurate however, the ANN model has been able to locate some hotspots.

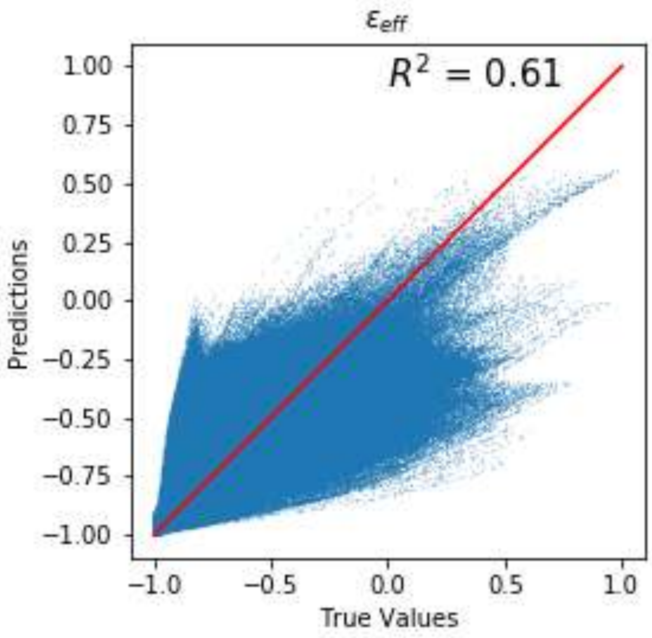


Figure 4.2: Plot of ANN predicted values vs FE values for effective strain trained alone

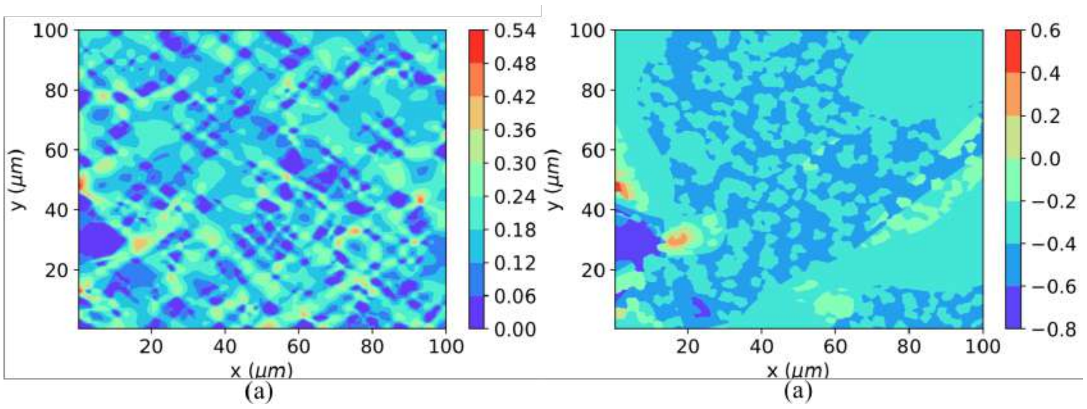


Figure 4.3: Effective Strain contour plots for (a) FE and (b) ANN predicted values.

4.2 LSTM Model Results

The LSTM models have first been used to predict the deformation in microstructures to which they have been trained and then they have been used to predict for the untrained microstructures. In this regard, multiple microstructures have been instantiated using the algorithm described in section 3.1.1. Specifically, the grain size ($d_{\alpha\prime}$) and volume fraction (V_α, $V_{\alpha\prime}$) of the constituent phases (ferrite, α, and martensite, $\alpha\prime$) have been varied parametrically. Further, five random microstructure instantiations are created for each set of microstructure variables. These set of microstructure variables are listed in table 4.2 and have been assigned distinct microstructure numbers for future reference. FE simulation results for these 25 microstructures serve as the ground truth for assessing the predictive capabilities of the LSTM models.

Table 4.2: Microstructural attributes of the different microstructures considered in the present work in terms of the grain sizes, d, and volume fractions, V, of the constituent phases, α and $\alpha\prime$.

Microstructure #	$d_{\alpha\prime}$	$V_{\alpha\prime}$	d_α	V_α
1,2,3,4,5	1.5	0.13	2.5	0.87
6,7,8,9,10	0.5	0.13	2.5	0.87
11,12,13,14,15	2.5	0.13	2.5	0.87
16,17,18,19,20	1.5	0.2	2.5	0.8
21,22,23,24,25	1.5	0.3	2.5	0.7

Different measures have been used to assess the error associated with the LSTM model predictions, as compared to the ground truth (FE results). They are briefly described in the following.

The coefficient of determination, R^2, represents the proportion of variance that has been explained by the independent variables in the model and is given by the following equation:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{m=1}^{n}(y_m - \hat{y}_m)^2}{\sum_{m=1}^{n}(y_m - \bar{y})^2} \tag{4.2}$$

where, y_m is the actual output (ground true value), and $\hat{y}_m$ is the model prediction. For a model with 100% accuracy, $R^2 = 1$. We also use the Mean Absolute Error (MAE) as an error metric, which is given by the equation:

$$MAE = \frac{\sum_{m=1}^{n} |y_m - \hat{y}_m|}{n} \tag{4.3}$$

4.2.1 Effect of Training Data on LSTM Model Results

Four models with the same architecture but different training data sets were trained and their predictions for all the 25 microstructures were compared. Model I was trained only with microstructure #1 and the results of this model are presented in the table 4.3. It can be seen that values of R^2 are generally low for most microstructures with only a few values above 0.90; R^2 values closer to 1 are desirable. The predictions for the first five microstructures are somewhat better than the other microstructures because their microstructural attributes are similar to the microstructure used in training. The higher accuracy in von Mises effective stress values can be attributed to a lower spread in stress values. Stress is low in the ferrite phase and high in the martensite phase, not a lot of intermediate values are present hence easier for the model to predict. However the prediction accuracy for the rest of the microstructures is not good, showing the model is unable to generalize over other data sets.

Model II has been trained with 2 microstructures, # 1 and # 21. As can be observed in table 4.4, the accuracy for microstructures 21-25 has increased compared to Model-I due to the inclusion of # 21 in the training dataset. The model is now able to deal with a few changes in microstructural parameters but still showing poor accuracy for rest of the microstructures. Even the accuracy for microstructures 1-5 has decreased a little. The next logical step is to add another dataset to the training data and see the results.

Table 4.5 shows the results for Model-III which has been trained with 3 microstructures, # 1, # 11 and # 21. We observe an overall improvement in accuracy of the model. The mean average error has reduced for all microstructures. However, even though it's performance is comparatively better than the previous model the R^2 values are still in the range of

TABLE 4.3: R^2 and MAE of LSTM model I predictions for the 25 microstructures, which was trained to microstructure # 1.

Microstructure	$\bar{\epsilon}$		$\bar{\sigma}$		ST	
	R^2	MAE	R^2	MAE (MPa)	R^2	MAE
1	0.88	0.012	0.99	31.303	0.90	0.072
2	0.85	0.012	0.98	34.025	0.92	0.052
3	0.72	0.017	0.90	51.129	0.81	0.0817
4	0.74	0.016	0.84	69.031	0.75	0.113
5	0.81	0.013	0.97	58.959	0.86	0.069
6	0.59	0.0214	0.91	62.380	0.76	0.100
7	0.81	0.0143	0.99	34.737	0.86	0.078
8	0.67	0.019	0.68	85.889	0.60	0.148
9	0.75	0.017	0.96	62.142	0.90	0.061
10	0.60	0.019	0.97	44.194	0.87	0.065
11	0.42	0.025	0.35	124.060	0.67	0.118
12	0.63	0.020	0.60	90.435	0.73	0.111
13	0.83	0.014	0.99	29.540	0.89	0.067
14	0.28	0.024	0.22	132.300	0.07	0.221
15	0.71	0.018	0.98	36.64	0.85	0.077
16	0.80	0.020	0.97	29.932	0.88	0.085
17	0.32	0.031	0.30	155.257	0.82	0.108
18	0.68	0.022	0.73	86.229	0.79	0.120
19	0.70	0.023	0.77	71.911	0.85	0.096
20	0.77	0.019	0.89	45.494	0.87	0.084
21	0.73	0.025	0.67	106.94	0.75	0.14
22	0.85	0.019	0..83	76.413	0.76	0.137
23	0.54	0.033	0.38	169.54	0.75	0.153
24	0.85	0.020	0.84	68.899	0.81	0.126
25	0.77	0.023	0.76	90.865	0.78	0.133

0.70-0.80. We now train the model with a microstructure of each type to account for every change in attribute.

Model IV has been trained with data from 5 microstructures, # 1, # 6, # 11, # 16 and # 21 to account for all the changes in microstructural attributes and improve the ability of the model to generalize over a wider data set. The only trade off in increasing the training data is the increase in training time. The accuracy as visible from table 4.6 increases as well, and the time to make predictions remains the same. All values here are greater than 0.80 with most values above 0.85. Only # 10 and # 17 have slighter lower R^2 in the range of 0.70. We further discuss the best and worst predictions made my this model on

Table 4.4: R^2 and MAE of LSTM model II predictions for the 25 microstructures, which was trained to microstructure # 1 and # 21.

Microstructure	$\bar{\epsilon}$		$\bar{\sigma}$		ST	
	R^2	MAE	R^2	MAE (MPa)	R^2	MAE
1	0.92	0.010	0.99	14.923	0.93	0.053
2	0.83	0.013	0.96	61.048	0.90	0.058
3	0.60	0.021	0.98	39.241	0.52	0.147
4	0.71	0.0176	0.96	60.185	0.42	0.179
5	0.83	0.014	0.94	77.031	0.86	0.073
6	0.52	0.025	0.86	123.281	0.64	0.11
7	0.77	0.017	0.92	96.019	0.80	0.904
8	0.71	0.020	0.91	99.270	0.53	0.144
9	0.78	0.017	0.95	74.293	0.87	0.067
10	0.65	0.019	0.91	93.678	0.83	0.073
11	0.59	0.030	0.85	100.121	0.39	0.120
12	0.67	0.020	0.92	89.124	0.46	0.156
13	0.77	0.017	0.93	88.653	0.80	0.091
14	0.28	0.021	0.49	65.094	-1.00	0.272
15	0.62	0.023	0.90	107.619	0.72	0.107
16	0.69	0.027	0.99	35.869	0.89	0.085
17	0.77	0.020	0.98	50.113	0.78	0.112
18	0.80	0.020	0.95	89.889	0.71	0.128
19	0.75	0.023	0.97	75.312	0.78	0.112
20	0.78	0.020	0.96	69.291	0.85	0.091
21	0.88	0.017	0.99	24.95	0.89	0.090
22	0.85	0.019	0.98	39.73	0.80	0.123
23	0.86	0.019	0.99	31.062	0.87	0.098
24	0.80	0.024	0.98	45.25	0.80	0.142
25	0.87	0.017	0.99	26.22	0.90	0.090

an untrained dataset.

4.2.2 Predictions of Model IV on Untrained Microstructure

The tables presented above all show the accuracy of the model in predicting the last strain step. However, the model is able to predict all the intermediate steps with reasonable accuracy as well. The error in predictions using LSTM model IV is graphically represented using figure 4.4. Figure 4.4 shows a box plot of the mean absolute error (MAE) values for the three output variables: (a) $\bar{\epsilon}$, (b) $\bar{\sigma}$, and (c) ST at every time step for all the 25 microstructures. This is plotted using the MAEs for all material points and all 15 time

Table 4.5: R^2 and MAE of LSTM model III predictions for the 25 microstructures, which was trained to microstructure # 1, # 11 and # 21.

Microstructure	$\bar{\epsilon}$		$\bar{\sigma}$		ST	
	R^2	MAE	R^2	MAE (MPa)	R^2	MAE
1	0.91	0.010	0.99	15.275	0.99	0.050
2	0.85	0.012	0.97	58.303	0.90	0.057
3	0.78	0.015	0.92	96.332	0.62	0.122
4	0.82	0.013	0.92	98.062	0.54	0.148
5	0.86	0.012	0.94	80.562	0.81	0.091
6	0.71	0.017	0.89	113.017	0.62	0.126
7	0.84	0.015	0.93	93.801	0.78	0.100
8	0.80	0.015	0.92	98.405	0.48	0.166
9	0.84	0.013	0.95	77.387	0.82	0.084
10	0.75	0.015	0.92	95.415	0.80	0.082
11	0.78	0.017	0.94	82.321	0.14	0.212
12	0.79	0.015	0.92	95.187	0.46	0.159
13	0.80	0.016	0.94	84.467	0.78	0.100
14	0.79	0.015	0.94	79.423	-1.09	0.35
15	0.79	0.016	0.92	97.222	0.71	0.115
16	0.72	0.025	0.99	28.74	0.90	0.076
17	0.85	0.017	0.98	44.013	0.82	0.103
18	0.84	0.017	0.96	76.644	0.71	0.139
19	0.81	0.019	0.97	65.508	0.78	0.116
20	0.82	0.018	0.97	61.784	0.83	0.100
21	0.88	0.017	0.99	22.74	0.89	0.085
22	0.85	0.020	0.98	54.581	0.80	0.121
23	0.88	0.019	0.99	29.250	0.87	0.100
24	0.82	0.024	0.98	55.185	0.85	0.113
25	0.89	0.017	0.99	24.954	0.90	0.087

steps for each of the three output variables. Essentially, this comprises $160,000 \times 15$ data points for each output variable for each microstructure.

In order to demonstrate the model's predictive capabilities, we present the deformation contours for a microstructure to which the model IV has not been trained. We choose microstructure # 14 and # 17 for this purpose. Microstructure # 14 shows the best results compared to other microstructures to which the model has not been trained, likewise, # 17 shows the worst. Figure 4.5 shows the predicted values vs true values plot of our worst prediction, # 17 and figure 4.6 contains our best prediction, # 14. It can be noticed that the spread of values of the $y = x$ line is much lower in 4.6(a) compared to 4.5(a). Similar

Table 4.6: R^2 and MAE of LSTM model IV predictions for the 25 microstructures, which was trained to microstructure # 1, # 6, # 11, # 16 and # 21.

Microstructure	$\bar{\epsilon}$		$\bar{\sigma}$		ST	
	R^2	MAE	R^2	MAE (MPa)	R^2	MAE
1	0.91	0.010	0.99	16.874	0.93	0.054
2	0.86	0.012	0.96	61.623	0.89	0.059
3	0.89	0.010	0.99	28.221	0.91	0.056
4	0.87	0.011	0.99	17.001	0.89	0.069
5	0.84	0.012	0.95	71.509	0.85	0.072
6	0.91	0.009	0.99	17.191	0.93	0.048
7	0.87	0.012	0.97	54.258	0.90	0.063
8	0.87	0.012	0.99	20.403	0.92	0.057
9	0.83	0.013	0.95	80.992	0.84	0.074
10	0.74	0.015	0.92	90.096	0.81	0.079
11	0.92	0.009	0.99	14.130	0.93	0.048
12	0.90	0.010	0.99	17.262	0.92	0.051
13	0.86	0.012	0.99	28.989	0.92	0.057
14	0.92	0.009	0.99	14.725	0.91	0.057
15	0.90	0.010	0.99	30.021	0.91	0.056
16	0.81	0.020	0.99	23.751	0.90	0.078
17	0.72	0.023	0.98	49.489	0.78	0.117
18	0.87	0.015	0.99	27.456	0.88	0.081
19	0.86	0.017	0.99	37.360	0.88	0.085
20	0.83	0.018	0.99	42.115	0.88	0.081
21	0.90	0.016	0.99	22.904	0.90	0.085
22	0.88	0.017	0.99	39.181	0.85	0.108
23	0.86	0.019	0.99	31.765	0.87	0.101
24	0.87	0.018	0.99	33.653	0.85	0.120
25	0.89	0.017	0.99	25.252	0.91	0.085

observation can be made in the prediction of stress triaxiality ratios as well.

Contour plots of LSTM predictions and the associated errors at the last strain step for our best prediction, microstructure # 14, are shown in figure 4.7(a), 4.8(a) and 4.9(a) for: $\bar{\epsilon}$, $\bar{\sigma}$, and ST, respectively. The corresponding errors are shown in figure 4.7(b), 4.8(b) and 4.9(b) for $\bar{\epsilon}$, $\bar{\sigma}$, and ST, respectively. It is clear from figure 4.7(a) and (b) that the LSTM model is able to predict the strain localization trends with reasonable accuracy. For example, while the maximum value of $\bar{\epsilon}$ is 0.40, the range of associated error for most of the microstructure is $[-0.060, 0.060]$. Similarly, Figure 4.8(b) shows that the error in prediction of $\bar{\sigma}$ is negligible. While the maximum $\bar{\sigma}$ is of the order of 1800 MPa, the scale for error contours lies in the range $[-80, 80]$ MPa. The ST contour and the associated error also demonstrates similar trends. It should also be noted that the deformation contours for this microstructure (# 14, $V_{\alpha'}$ of 0.20) are quite distinct from that presented in figure 3.2 for microstructure # 1, with $V_{\alpha'}$ of 0.13. Although the strain localizations and stress hot spots appear very distinct in these two microstructures, the LSTM model gives accurate predictions of the deformation contours in the untrained microstructure. Generally speaking, the LSTM model IV is able to predict the heterogeneous deformation contours in an untrained microstructure with high accuracy.

We have also used the LSTM model to predict the strain contours at intermediate strain steps. The LSTM predicted contours of $\bar{\epsilon}$ are shown in figure 4.10(a) and 4.11(a) for 0.05 and 0.10 applied strain, respectively, along with the associated errors in figure 4.10(b) and 4.11(b), respectively. Note that although the strain localization patterns appear similar at both these applied strains, the actual values of strains are evolving and distinct. It can be seen from the error contours at both these applied strains that the associated errors are relatively low. These results indicate the LSTM model's capability to predict the spatio-temporal evolution of heterogeneous deformation fields in dual phase microstructures with high accuracy.

Finally, we have used the LSTM predicted contours to calculate the average values of $\bar{\sigma}$ and $\bar{\epsilon}$ at each strain step. These are then used to re-construct the ensemble-average stress-strain response. Figure 4.12(a) shows a comparison of the LSTM predicted stress-strain response with the 'ground truth' FE stress-strain response. It can be seen that the LSTM response compares reasonably with the FE response.

Strain partitioning is commonly observed in these dual phase microstructures. This occurs due to higher deformation being accommodated in the softer phase. On the other hand, the harder phase, with a relatively higher yield stress, does not accommodate much deformation and primarily contributes to strengthening of the microstructure. Figure 4.12(b) shows the comparison of the LSTM predicted and FE predicted average effective strains in the soft and the hard phases as a function of applied strain. It can be seen that the LSTM model is under-predicting the average strain in the soft phase at higher applied strains, as compared to the FE model. Prediction of average effective strain in the hard phase appears to be reasonably accurate. The strain partitioning effect due to the presence of a hard and soft phase in the dual phase microstructure is clearly visible from these plots. Underlying mechanisms are discussed in more detail in the next section.

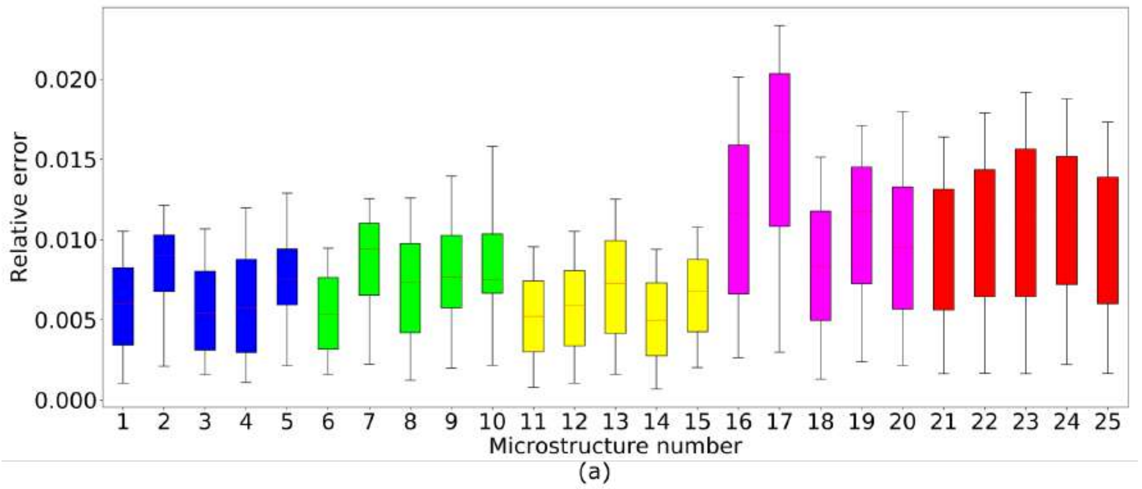
4.3 Discussion

The application problem chosen in this work is the elastic-plastic deformation of a two phase microstructure. This microstructure is representative of Dual Phase (DP) steels, comprised of the soft ferrite phase and the relatively harder martensite phase. Since there is a large strength differential between the phases (> 1 GPa), the stress is usually partitioned to the hard martensite phase, while the strain is partitioned to the softer ferrite phase. This was also observed in our simulations (cf. figure 4.12(b)). At the early stages of deformation, the ferrite phase starts to deform plastically due to its lower yield strength, while the martensite still deforms elastically. At the later stages, the martensite phase also starts to show some plastic deformation, although with limited strain hardening [56, 57] due to its complex hierarchical structure [58, 59]. As the ferrite phase deforms plastically, accumulation of dislocation forests lead to large amounts of strain hardening, especially in the ferrite grains lying between closely spaced martensite islands. Eventually, the strength differential between the two phases is reduced and the strain also starts to partition to the martensite phase [60, 61]. Severe strain localization occurs, especially at the corners of sharp martensite islands, which act as hot spots for eventual nucleation of damage [57, 61].

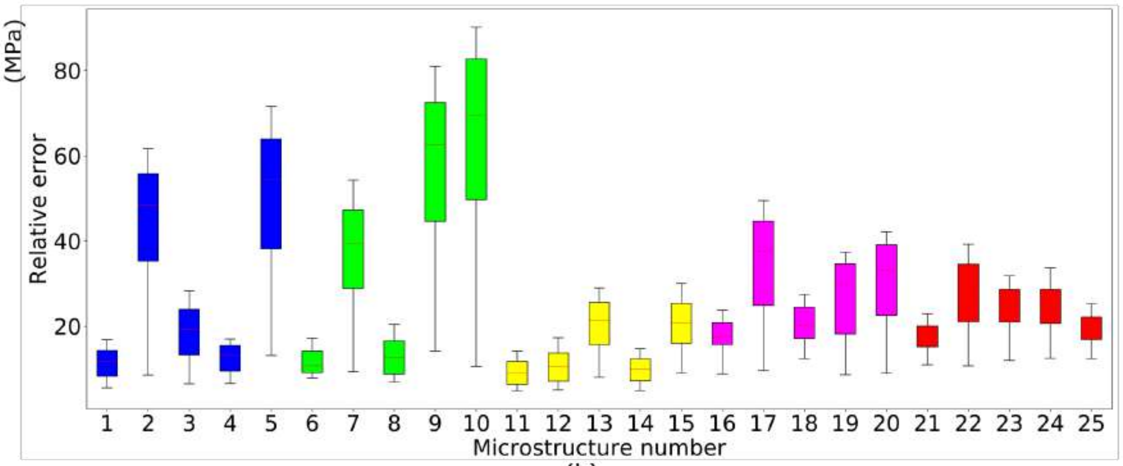
It is therefore important to predict such hot spots for large microstructures using computationally efficient techniques, such as ML-based surrogate models. Predicting the same for large dual phase microstructures, with heterogeneous deformation characteristics, may perhaps be too computationally expensive for physics-based FE frameworks. As a comparison, each of the FE simulations took ≈ 1440 compute hours (60 physical hours for a simulation running on 24 core Intel Xeon processors and 4 GB memory per core) to reach

0.15 applied strain. In contrast, the LSTM model IV, with 8 layers and 100 neurons in each layer, was trained in less than $\approx$ 112 compute hours (3.5 physical hours for a training running on 8 core Intel i7 processor and 2 GB memory per core. Once trained, the LSTM model IV was able to predict the deformation contours in less than 2 mins on the same workstation computer. This amounts to a speed up of $\approx$ 700 times as compared to the FE simulations, excluding the training time (which only needs to be performed once).

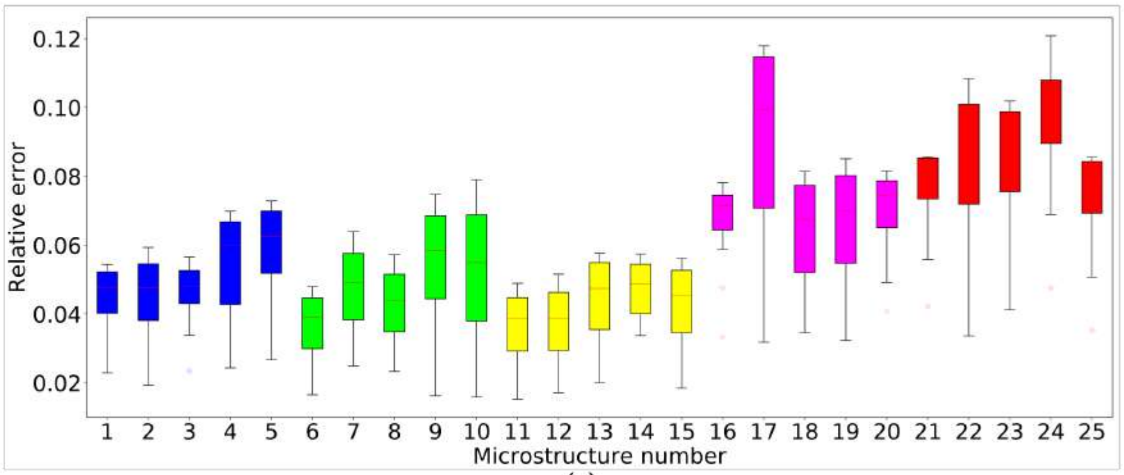
The LSTM model IV predicts all three output variables with reasonable quantitative accuracy for trained as well untrained microstructures. Plastic deformation being a history-dependent process, this behavior is better modeled using LSTM networks, as they treat the data as a time series. LSTMs maintain a context over time, thus making them efficient in time series predictive modeling problems. Pandey and Pokharel [29] used a similar approach to predict the evolution of crystallographic orientation in polycrystalline ensembles undergoing plastic deformation. Their model also considers data from the nearest neighbourhood to predict the orientation evolution at any material point. In comparison, our LSTM model is much simpler and does not consider the effect of neighbors. Logarzo et al. [34] also used sequence to sequence modeling to develop Smart Constitutive Laws (SCLs) to predict the response of any material given the loading histories. They generate a number strain sequences which are used as inputs to predict the stress response. Their model establishes a relationship between the strain sequences and the corresponding stresses, thus acting as an 'effective' constitutive law. In contrast, our approach predicts all variables independently only using their individual histories as inputs.



(a)



(b)



(c)

FIGURE 4.4: Box plots of mean squared errors (MSEs) for LSTM model IIa predicted values for (a) $\bar{\epsilon}$, (b) $\bar{\sigma}$, and (c) ST across all material points and all time steps for the different microstructures. The different colors in the box plots are used to denote the different microstructure attributes considered in table 4.2.

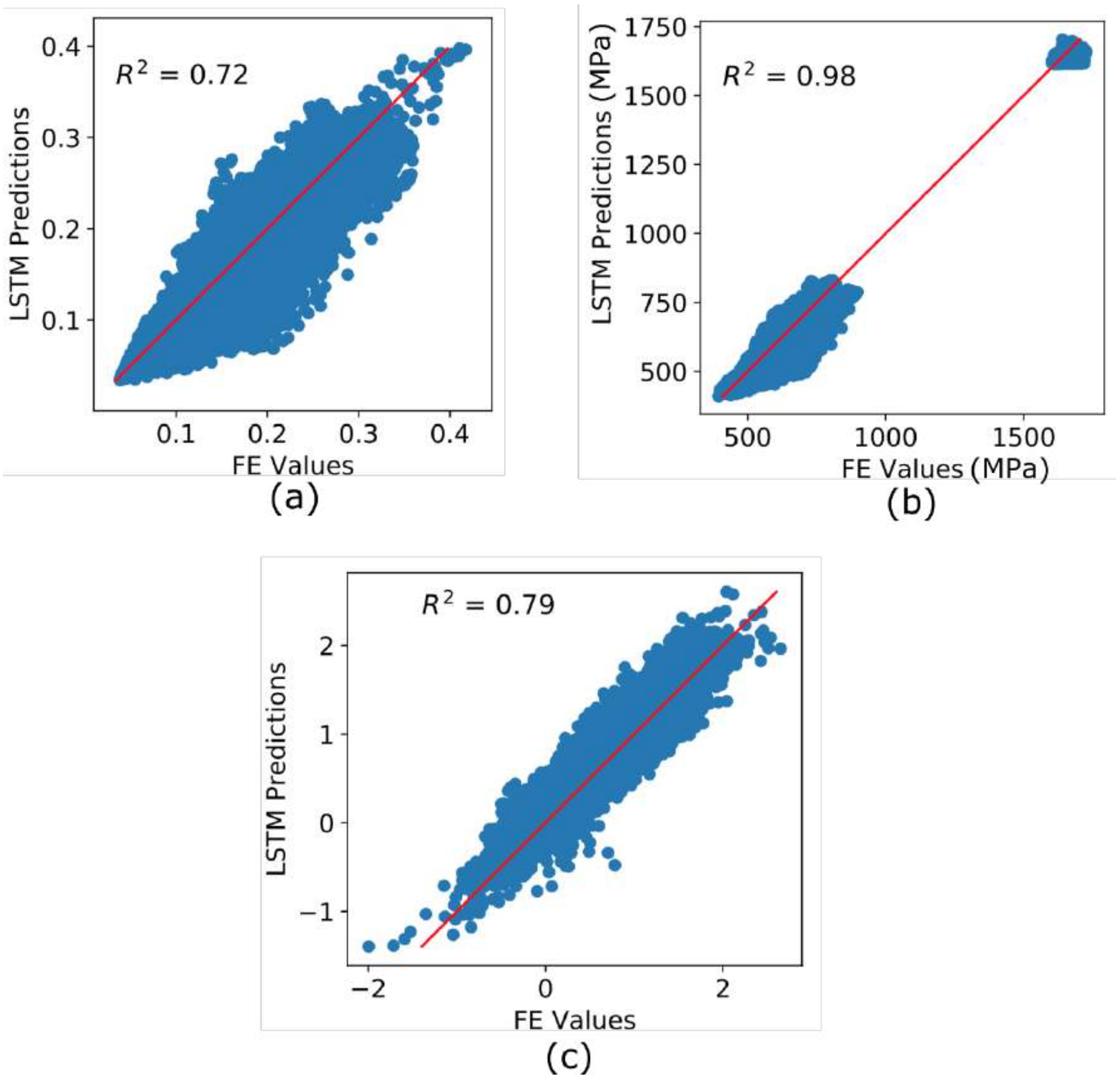
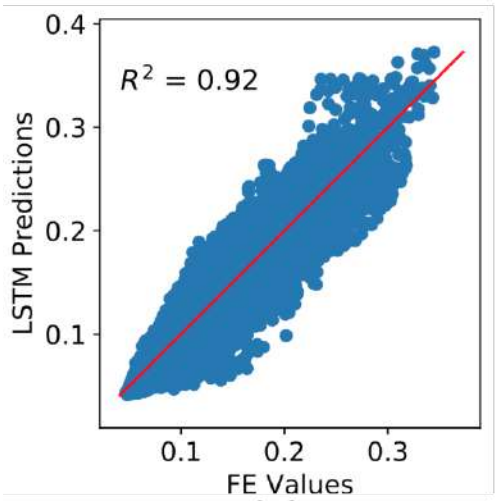
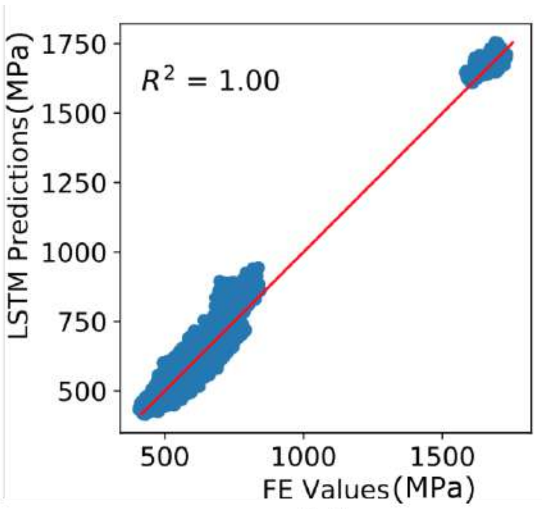


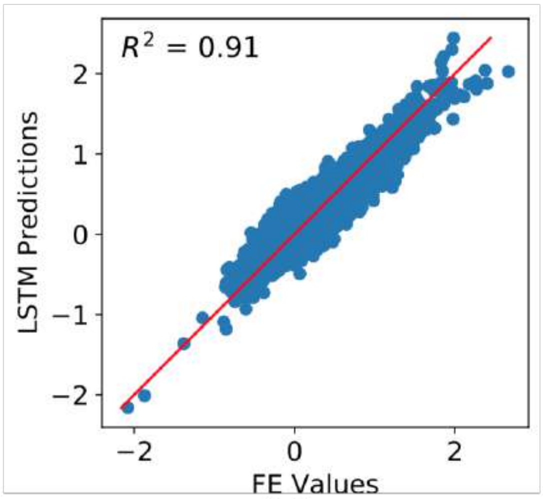
FIGURE 4.5: Plot of LSTM predicted values vs FE values for (a) effective strain, (b) von Mises effective stress, (c) triaxiality for microstructure # 17



(a)



(b)



(c)

FIGURE 4.6: Plot of LSTM predicted values vs FE values for (a) effective strain, (b) von Mises effective stress, (c) triaxiality for microstructure # 14

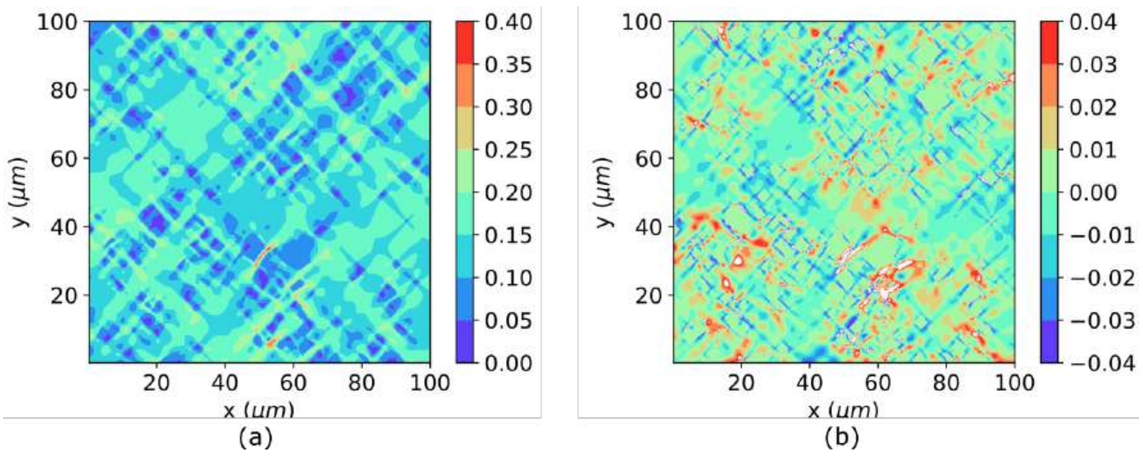


FIGURE 4.7: Effective Strain contour plots for (a) LSTM predicted values and (b) Error in predictions at the last time step for microstructure # 14.

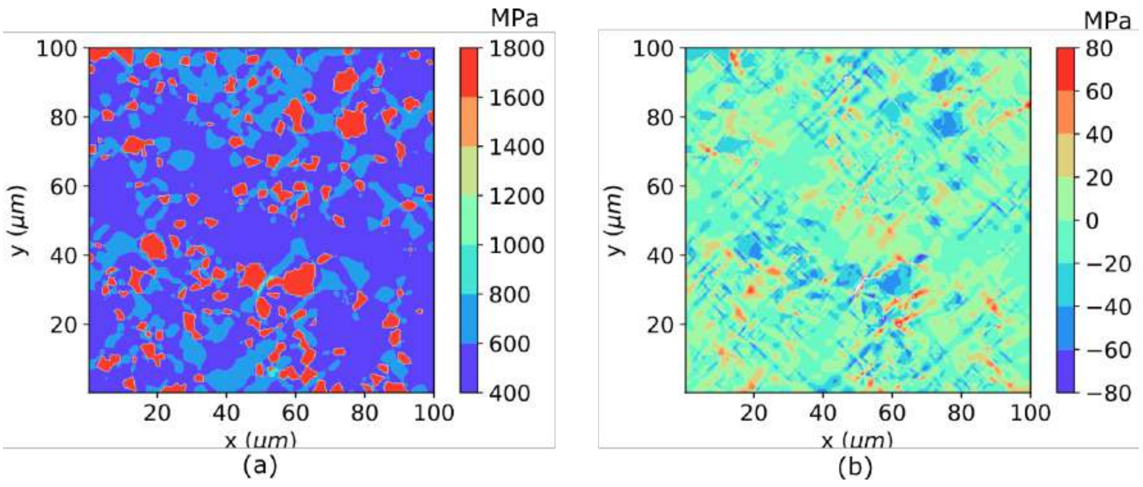


FIGURE 4.8: Vonmises contour plots for (a) LSTM predicted values and (b) Error in predictions at the last time step for microstructure # 14.

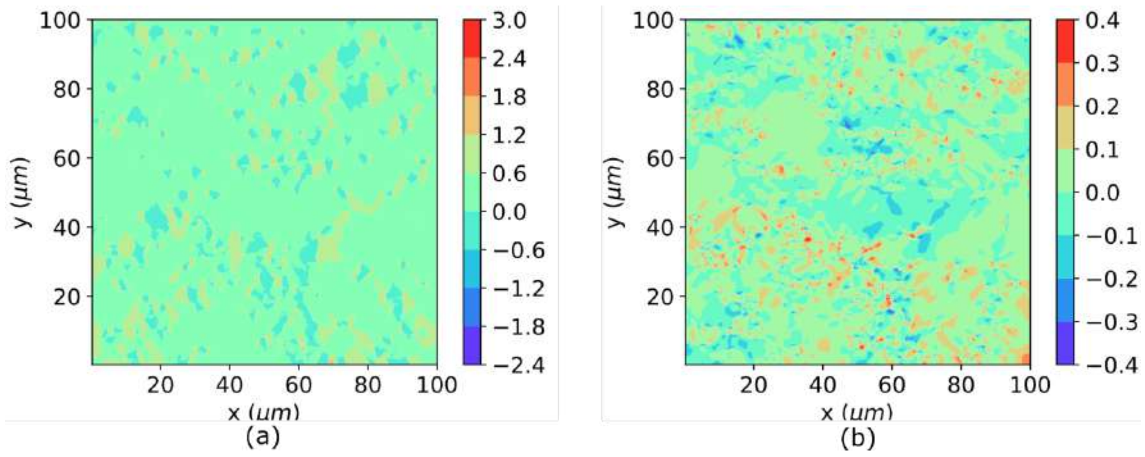


Figure 4.9: Triaxiality contour plots for (a) LSTM predicted values and (b) Error in predictions at the last time step for microstructure # 14.

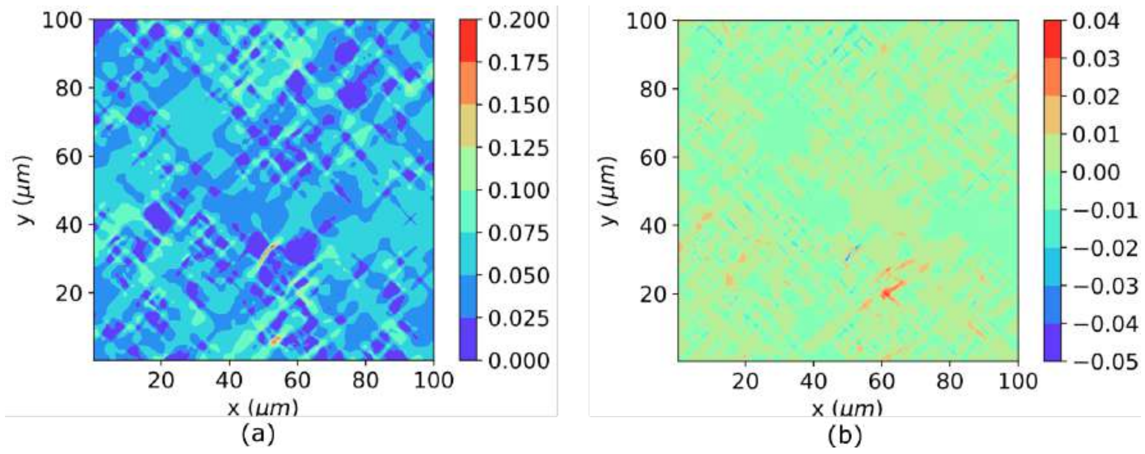


Figure 4.10: Spatial contours of LSTM model IV predicted values for $\bar{\epsilon}$ at (a) 0.05 applied strain for microstructure # 14. The corresponding error contours are plotted in (b)

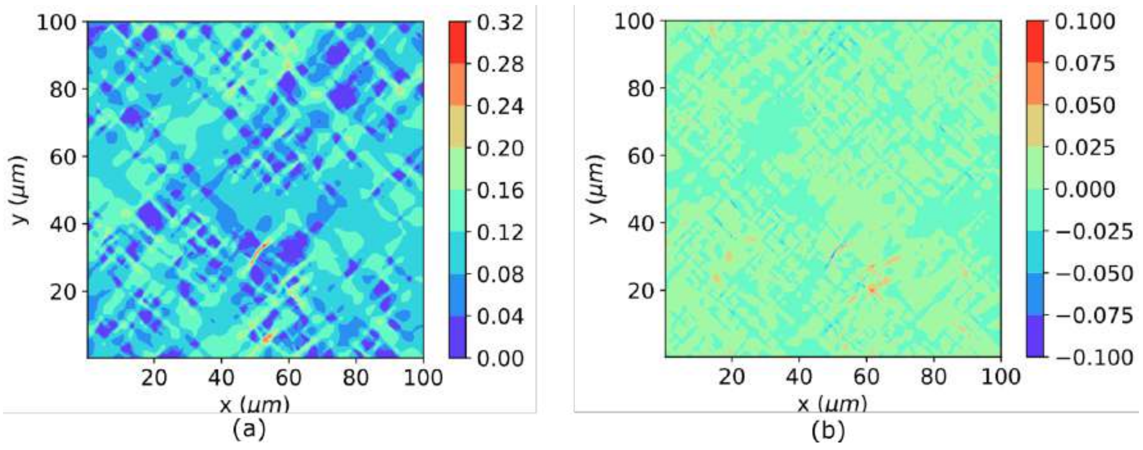


FIGURE 4.11: Spatial contours of LSTM model IV predicted values for $\bar{\epsilon}$ at (a) 0.10 applied strain for microstructure # 14. The corresponding error contours are plotted in (b)

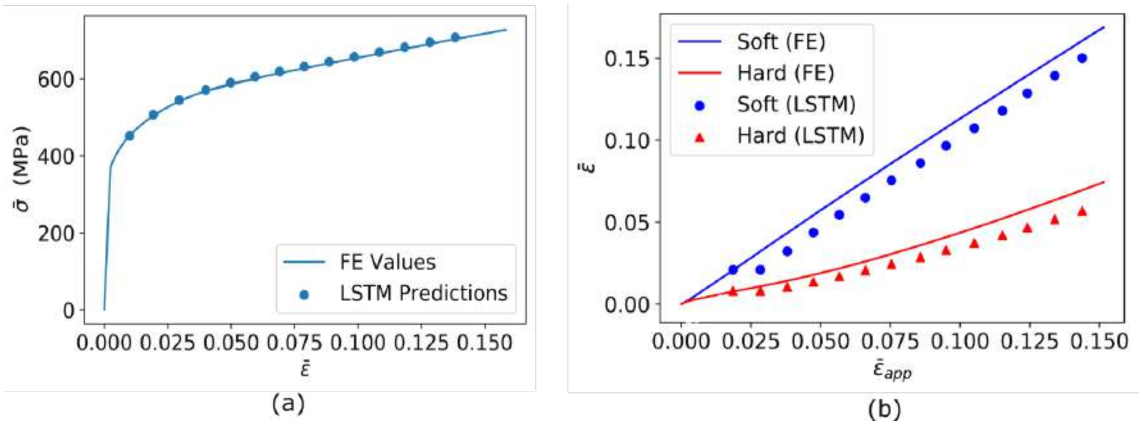


FIGURE 4.12: (a) Comparison of re-constructed stress-strain curve from LSTM model IV predictions with the FE results for microstructure # 14. (b) Average effective strains in the soft and hard phases as a function of applied strain as predicted using the LSTM model IV and the FE model for the same microstructure.

Chapter 5

Conclusion

5.1 Summary

We have applied two machine learning models, namely, artificial neural networks (ANNs) and long short term memory (LSTM), a type of recurrent neural network (RNN) to predict the deformation of DP steel microstructures. J2 plasticity simulations are run for the 2D dual-phase ferrite-martensite microstructures. The results of these simulations are used as the ground truth for the machine learning model training. We start with employing an ANN to make predictions for effective strain, von Mises effective stress and triaxiality. After a systematic parametric study for the hyper parameters of the model, the accuracy of the ANN models was found to be very low. ANNs do not deal with temporal or spatial information. The higher errors may also be a result of discretizing our dataset into 0.01 strain steps.

As plasticity is a history-dependent process, RNNs are presumably a better option. LSTMs were applied to deal with the temporal nature of our data. The results obtained from LSTM model have an error value 3 order less than that of the ANN. The model makes predictions at 0.01 strain intervals. Our method does not need to perform numerical iterations per strain step, otherwise needed by the conventional methods. The main strength of the LSTM model over ANN, is the reduction in computational time. The training time is 7 to 8 times less than that for ANN and predictions only take a few seconds. This ML-surrogate model trained and tested on simulated data is highly accurate and orders of magnitude faster than conventional micromechanical models.

5.2 Future Work

The models developed is capable of predicting the evolution of a microstructure given a first initial step, that is, given the values at 0.01 strain our model can successfully evolve the microstructure. We ideally want our model to predict the 1st step as well by looking at just the phase map of the microstructure.

Bibliography

[1] W. Muhammad, A. P. Brahme, O. Ibragimova, J. Kang, and K. Inal, "A machine learning framework to predict local strain distribution and the evolution of plastic anisotropy & fracture in additively manufactured alloys," *International Journal of Plasticity*, p. 102867, 2020.

[2] *Applications of Machine learning*, 2018 (accessed October 15, 2020). https://www.javatpoint.com/applications-of-machine-learning.

[3] M. Knezevic, J. Crapps, I. J. Beyerlein, D. R. Coughlin, K. D. Clarke, and R. J. McCabe, "Anisotropic modeling of structural components using embedded crystal plasticity constructive laws within finite elements," *International Journal of Mechanical Sciences*, vol. 105, pp. 227 – 238, 2016.

[4] A. Patra and C. N. Tomé, "Finite element simulation of gap opening between cladding tube and spacer grid in a fuel rod assembly using crystallographic models of irradiation growth and creep," *Nuclear Engineering and Design*, vol. 315, pp. 155–169, 2017.

[5] F. Roters, P. Eisenlohr, L. Hantcherli, D. D. Tjahjanto, T. R. Bieler, and D. Raabe, "Overview of constitutive laws, kinematics, homogenization and multiscale methods in crystal plasticity finite-element modeling: Theory, experiments, applications," *Acta Materialia*, vol. 58, no. 4, pp. 1152–1211, 2010.

[6] K. Thool, A. Patra, D. Fullwood, K. M. Krishna, D. Srivastava, and I. Samajdar, “The role of crystallographic orientations on heterogeneous deformation in a zirconium alloy: A combined experimental and modeling study,” *International Journal of Plasticity*, vol. 133, p. 102785, 2020.

[7] C. Mayer, L. Yang, S. Singh, J. Llorca, J. Molina-Aldareguia, Y. Shen, and N. Chawla, “Anisotropy, size, and aspect ratio effects on micropillar compression of alsic nanolaminate composites,” *Acta Materialia*, vol. 114, pp. 25–32, 2016.

[8] J. Knapp, D. Follstaedt, S. Myers, J. Barbour, and T. Friedmann, “Finite-element modeling of nanoindentation,” *Journal of Applied Physics*, vol. 85, no. 3, pp. 1460–1474, 1999.

[9] N. Zaafarani, D. Raabe, R. N. Singh, F. Roters, and S. Zaefferer, “Three-dimensional investigation of the texture and microstructure below a nanoindent in a cu single crystal using 3d ebsd and crystal plasticity finite element simulations,” *Acta Materialia*, vol. 54, no. 7, pp. 1863–1876, 2006.

[10] D. L. McDowell, “Viscoplasticity of heterogeneous metallic materials,” *Materials Science and Engineering: R: Reports*, vol. 62, no. 3, pp. 67–123, 2008.

[11] O. Sedaghat and H. Abdolvand, “A non-local crystal plasticity constitutive model for hexagonal close-packed polycrystals,” *International Journal of Plasticity*, vol. 136, p. 102883, 2021.

[12] M. R. Tonks, D. Gaston, P. C. Millett, D. Andrs, and P. Talbot, “An object-oriented finite element framework for multiphysics phase field simulations,” *Computational Materials Science*, vol. 51, no. 1, pp. 20–29, 2012.

[13] S. Lee, M. F. Wheeler, and T. Wick, “Pressure and fluid-driven fracture propagation in porous media using an adaptive finite element phase field model,” *Computer Methods in Applied Mechanics and Engineering*, vol. 305, pp. 111–132, 2016.

[14] S. Na and W. Sun, “Computational thermomechanics of crystalline rock, part i: A combined multi-phase-field/crystal plasticity approach for single crystal simulations,”

Computer Methods in Applied Mechanics and Engineering, vol. 338, pp. 657–691, 2018.

[15] C. Liu, P. Shanthraj, M. Diehl, F. Roters, S. Dong, J. Dong, W. Ding, and D. Raabe, "An integrated crystal plasticity–phase field model for spatially resolved twin nucleation, propagation, and growth in hexagonal materials," *International Journal of Plasticity*, vol. 106, pp. 203–227, 2018.

[16] R. A. Lebensohn, A. K. Kanjarla, and P. Eisenlohr, "An elasto-viscoplastic formulation based on fast fourier transforms for the prediction of micromechanical fields in polycrystalline materials," *International Journal of Plasticity*, vol. 32-33, pp. 59 – 69, 2012.

[17] R. A. Lebensohn and A. Needleman, "Numerical implementation of non-local polycrystal plasticity using fast fourier transforms," *Journal of the Mechanics and Physics of Solids*, vol. 97, pp. 333–351, 2016.

[18] A. Marano, L. Gélébart, and S. Forest, "Fft-based simulations of slip and kink bands formation in 3d polycrystals: influence of strain gradient crystal plasticity," *Journal of the Mechanics and Physics of Solids*, p. 104295, 2021.

[19] C. C. Tasan, M. Diehl, D. Yan, C. Zambaldi, P. Shanthraj, F. Roters, and D. Raabe, "Integrated experimental–simulation analysis of stress and strain partitioning in multiphase alloys," *Acta Materialia*, vol. 81, pp. 386–400, 2014.

[20] R. Pokharel, A. Patra, D. W. Brown, B. Clausen, S. C. Vogel, and G. T. Gray, "An analysis of phase stresses in additively manufactured 304l stainless steel using neutron diffraction measurements and crystal plasticity finite element simulations," *International Journal of Plasticity*, vol. 121, pp. 201 – 217, 2019.

[21] E. Ford, K. Maneparambil, S. Rajan, and N. Neithalath, "Machine learning-based accelerated property prediction of two-phase materials using microstructural descriptors and finite element analysis," *Computational Materials Science*, vol. 191, p. 110328, 2021.

[22] S. Hashemi and S. R. Kalidindi, "A machine learning framework for the temporal evolution of microstructure during static recrystallization of polycrystalline materials simulated by cellular automaton," *Computational Materials Science*, vol. 188, p. 110132, 2021.

[23] A. Beniwal, R. Dadhich, and A. Alankar, "Deep learning based predictive modeling for structure-property linkages," *Materialia*, vol. 8, p. 100435, 2019.

[24] R. Liu, Y. C. Yabansu, A. Agrawal, S. R. Kalidindi, and A. N. Choudhary, "Machine learning approaches for elastic localization linkages in high-contrast composite materials," *Integrating Materials and Manufacturing Innovation*, vol. 4, no. 1, pp. 192–208, 2015.

[25] A. Agrawal and A. Choudhary, "Perspective: Materials informatics and big data: Realization of the "fourth paradigm" of science in materials science," *Apl Materials*, vol. 4, no. 5, p. 053208, 2016.

[26] R. Liu, Y. C. Yabansu, Z. Yang, A. N. Choudhary, S. R. Kalidindi, and A. Agrawal, "Context aware machine learning approaches for modeling elastic localization in three-dimensional composite microstructures," *Integrating Materials and Manufacturing Innovation*, vol. 6, no. 2, pp. 160–171, 2017.

[27] A. Mangal and E. A. Holm, "Applied machine learning to predict stress hotspots i: Face centered cubic materials," *International Journal of Plasticity*, vol. 111, pp. 122 – 134, 2018.

[28] X. Li, Z. Liu, S. Cui, C. Luo, C. Li, and Z. Zhuang, "Predicting the effective mechanical property of heterogeneous materials by image based modeling and deep learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 347, pp. 735–753, 2019.

[29] A. Pandey and R. Pokharel, "Machine learning enabled surrogate crystal plasticity model for spatially resolved 3d orientation evolution under uniaxial tension," 2020.

[30] A. Pandey and R. Pokharel, "Machine learning based surrogate modeling approach for mapping crystal deformation in three dimensions," *Scripta Materialia*, vol. 193, pp. 1–5, 2021.

[31] A. Gupta, A. Cecen, S. Goyal, A. K. Singh, and S. R. Kalidindi, "Structure–property linkages using a data science approach: application to a non-metallic inclusion/steel composite system," *Acta Materialia*, vol. 91, pp. 239–254, 2015.

[32] T. Kirchdoerfer and M. Ortiz, "Data-driven computational mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 304, pp. 81–101, 2016.

[33] A. E. Tallman, M. A. Kumar, A. Castillo, W. Wen, L. Capolungo, and C. N. Tomé, "Data-driven constitutive model for the inelastic response of metals: Application to 316h steel," *Integrating Materials and Manufacturing Innovation*, vol. 9, no. 4, pp. 339–357, 2020.

[34] H. J. Logarzo, G. Capuano, and J. J. Rimoli, "Smart constitutive laws: Inelastic homogenization through machine learning," *Computer Methods in Applied Mechanics and Engineering*, vol. 373, p. 113482, 2021.

[35] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[36] D. Morgan and R. Jacobs, "Opportunities and challenges for machine learning in materials science," *Annual Review of Materials Research*, vol. 50, pp. 71–103, 2020.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[38] T. Tiwari, T. Tiwari, and S. Tiwari, "How artificial intelligence, machine learning and deep learning are radically different?," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 8, p. 1, 03 2018.

[39] N. Buduma and N. Locascio, *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms.* " O'Reilly Media, Inc.", 2017.

[40] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[41] J. Brownlee, "Why training a neural network is hard," 2019 (accessed October 10, 2020). https://machinelearningmastery.com/why-training-a-neural-network-is-hard/.

[42] R. Parmar, "Common loss functions in machine learning," 2018 (accessed October 8, 2020). https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[43] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? a new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[44] X. Wang, Y. Hua, E. Kodirov, and N. M. Robertson, "Imae for noise-robust learning: Mean absolute error does not treat examples equally and gradient magnitude's variance matters," 2020.

[45] M. A. Nielsen, *Neural networks and deep learning*, vol. 2018. Determination press San Francisco, CA, 2015.

[46] J. Brownlee, "Understand the impact of learning rate on neural network performance," 2019 (accessed October 13, 2020). https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

[47] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[48] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[49] A. Patra, M. W. Priddy, and D. L. McDowell, "Modeling the effects of microstructure on the tensile properties and micro-fracture behavior of mo–si–b alloys at elevated temperatures," *Intermetallics*, vol. 64, pp. 6–17, 2015.

[50] C. J. Permann, D. R. Gaston, D. Andrš, R. W. Carlsen, F. Kong, A. D. Lindsay, J. M. Miller, J. W. Peterson, A. E. Slaughter, R. H. Stogner, and R. C. Martineau, "MOOSE: Enabling massively parallel multiphysics simulation," *SoftwareX*, vol. 11, p. 100430, 2020.

[51] B. J. I. S. Soudip Basu, Anirban Patra, "A computational modelling framework for studying the microstructural factors inducing strain partitioning in dual phase steels,"

[52] W. C. Mills-Curran, A. P. Gilkey, and D. P. Flanagan, "Exodus: A finite element file format for pre-and postprocessing," tech. rep., Sandia National Labs., Albuquerque, NM (USA), 1988.

[53] G. Sjaardema, "Sandia engineering analysis code access system (seacas)," 2020(accessed June 5, 2020). https://github.com/gsjaardema/seacas.

[54] A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial intelligence*, vol. 97, no. 1-2, pp. 245–271, 1997.

[55] R. J. May, H. R. Maier, and G. C. Dandy, "Data splitting for artificial neural networks using som-based stratified sampling," *Neural Networks*, vol. 23, no. 2, pp. 283–294, 2010.

[56] R. Rodriguez and I. Gutiérrez, "Unified formulation to predict the tensile curves of steels with different microstructures," in *Materials Science Forum*, vol. 426, pp. 4525–4530, Trans Tech Publications Ltd., Zurich-Uetikon, Switzerland, 2003.

[57] J. Kadkhodapour, A. Butz, S. Ziaei-Rad, and S. Schmauder, "A micro mechanical study on failure initiation of dual phase steels under tension using single crystal plasticity model," *International Journal of Plasticity*, vol. 27, no. 7, pp. 1103–1125, 2011.

[58] T. Maki, "Morphology and substructure of martensite in steels," *Phase transformations in steels*, pp. 34–58, 2012.

[59] H. Bhadeshia and R. Honeycombe, *Steels: microstructure and properties.* Butterworth-Heinemann, 2017.

[60] J. Kang, Y. Ososkov, J. D. Embury, and D. S. Wilkinson, "Digital image correlation studies for microscopic strain distribution and damage in dual phase steels," *Scripta Materialia*, vol. 56, no. 11, pp. 999–1002, 2007.

[61] H. Ghadbeigi, C. Pinna, and S. Celotto, "Failure mechanisms in dp600 steel: Initiation, evolution and fracture," *Materials Science and Engineering: A*, vol. 588, pp. 420–431, 2013.

[62] P. R. Dawson, "Computational crystal plasticity," *International Journal of Solids and Structures*, vol. 37, no. 1, pp. 115 – 130, 2000.

[63] J. Mayeur and D. McDowell, "A three-dimensional crystal plasticity model for duplex ti–6al–4v," *International Journal of Plasticity*, vol. 23, no. 9, pp. 1457 – 1485, 2007.

[64] B. Liu, D. Raabe, F. Roters, P. Eisenlohr, and R. A. Lebensohn, "Comparison of finite element and fast fourier transform crystal plasticity solvers for texture prediction," *Modelling and Simulation in Materials Science and Engineering*, vol. 18, p. 085005, oct 2010.

[65] F. Roters, P. Eisenlohr, L. Hantcherli, D. Tjahjanto, T. Bieler, and D. Raabe, "Overview of constitutive laws, kinematics, homogenization and multiscale methods in crystal plasticity finite-element modeling: Theory, experiments, applications," *Acta Materialia*, vol. 58, no. 4, pp. 1152 – 1211, 2010.

[66] B. Radhakrishnan, G. Sarma, H. Weiland, and P. Baggethun, "Simulations of deformation and recrystallization of single crystals of aluminium containing hard particles," *Modelling and Simulation in Materials Science and Engineering*, vol. 8, pp. 737–750, aug 2000.

[67] M. F. Horstemeyer and D. J. Bammann, "Historical review of internal state variable theory for inelasticity," *International Journal of Plasticity*, vol. 26, no. 9, pp. 1310–1334, 2010.

[68] Y.-F. Shen, R. Pokharel, T. J. Nizolek, A. Kumar, and T. Lookman, "Convolutional neural network-based method for real-time orientation indexing of measured electron backscatter diffraction patterns," *Acta Materialia*, vol. 170, pp. 118–131, 2019.

[69] N. Kotkunde, A. D. Deole, and A. K. Gupta, "Prediction of forming limit diagram for ti-6al-4v alloy using artificial neural network," *Procedia materials science*, vol. 6, pp. 341–346, 2014.