**DEVELOPMENT OF A DEEP LEARNING ALGORITHM TO PREDICT FATIGUE LIFE**

1. Introduction:
2. Work Plan:
3. Database Design:
4. Neural Network:
   1. Layer design:
   2. Activation functions:
   3. Loss function:
5. Using crack growth prediction for fatigue life calculation:
6. Results so far(till mid-term evaluation)
7. Future work that remained due to COVID-19 pandemic:
8. **Introduction**

Fatigue life of a component or material plays a very important part in designing in mechanical engineering. Fatigue life is calculated from the crack growth rate(in mm per cycle). There are many methods to compute the crack growth rate with the relationship between crack growth and stress intensity factor range(∆K), but as seen from the experiments crack growths rate does not show linear relationship with stress intensity factor range(∆K). Non-linearity of da/dN-delta K curve limits the computation accuracy of physical formula base methods that assume the linearity or a rigid da/dN-∆K relationship. In this project we explore a more flexible approach towards the crack growth rate prediction, deep learning algorithms. Neural networks(NN) are capable of capturing extreme non-linearity in the data and multivariable dependencies, which makes NNs a considerable solution for such problems.

1. **Work Plan**

Our work starts with the data collection and dataset preparation, first for training our neural network; we have mined online available data and prepared training and testing datasets

We design a neural network for predicting the crack growth rate, after training our NN we do a comparison study of the performance of neural network and the classical methods being used.

After reaching a good neural network parameter combination, we design a model for calculating the fatigue life with the predicted crack growth rate. For establishing a solid statement on ability of our NN, we have to get experimental data of our own from laboratory experiments and do the process of training and testing on this newly collected data and compare the experimental and NN predicted fatigue life (in number of cycles). We can provide a tuning of NN parameters in the “*training and testing on crack growth rate prediction*” phase for better results from fatigue life calculation model.

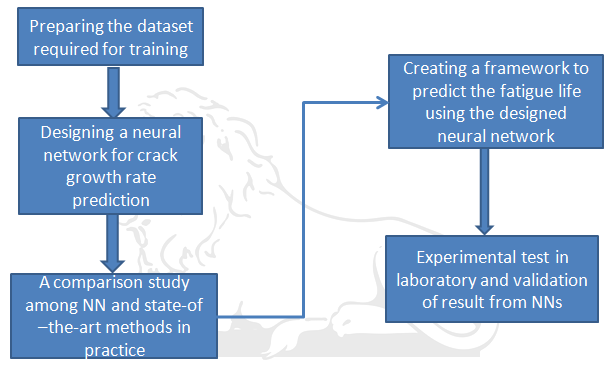


Figure 1. Work Plan

1. **Database design:**

We are using a 2024-T351 aluminum alloy data [2], this was originally created for a similar study on predicting fatigue life. With this data we have created a dataset with 3 data fields crack growth rate(da/dN), stress ratio(R) and stress intensity factor range(delta K) . For 4 different stress ratio values (0, 0.1, 0.3 and 0.5) we have 76 data points (da/dN – delta K pairs). Crack growth rate values are in order of 10-6, so for training we have multiplied crack growth rate column of the data with 105 for precision and avoiding over fitting. We train the neural network on the data of 3 stress ration (R) values and remaining data of 4th R value becomes our testing dataset. The reason for this type of training and testing is to see how our neural net can predict the crack growth rate for a stress ratio value that it has never seen during training, this also shows NN’s advantage over other classical fatigue crack growth rate calculation methods that has stress ratio(R) dependent prediction parameters. This data was mined using Origin software to get the graphical data.

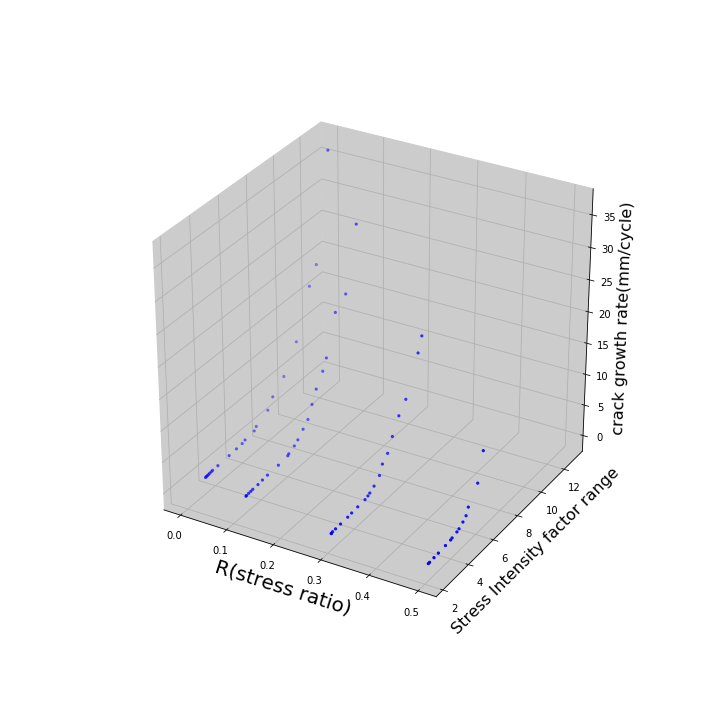


Figure 2. T351 aluminum alloy experimental data

1. **Neural network**

Neural Networks are specific set of machine learning algorithms inspired by biological neural networks. Like biological neural network, we sense input from outside and reacts in the form of outputs accordingly. Neural networks themselves are general function approximation and involve extremely data processing through extremely complex neuron circuits, which is why they can be applied to any machine learning problem that requires complex input-output space mapping. We in our project designed a standard neural network, with the following structure:

Output

W3

W2

W1

X3

X2

X1

Y

Figure 4.1: Structure of a single neuron in our network

Figure 4.1 shows the functionality of a single node or neuron in our network, each neuron takes input (Xi) from the active neurons of its previous layer. And a value Y is computed with the linear combination of input values with their corresponding weight coefficients. Line connecting two neurons is called connection and each connection has its own weight value. The output of each neuron is calculated after passing the computed Y through an activation function, activation functions of our neural network shall be discussed in next section.

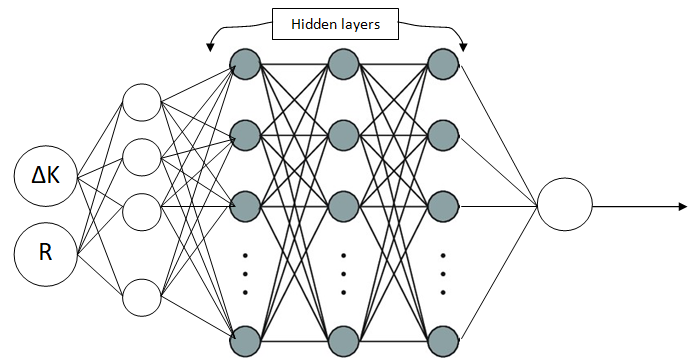


Figure 4.2: Layer structure of the neural network

4.1 Layer structure of our Neural Network:

Each neural network can be broken into three type of neuron layers, named as input layer, output layer and hidden layers, Layer 1 is our input layer, instead of (∆K-R) layer. As we have only two input parameters, we are exploding these two parameters and creating an input layer of 128 neurons, which is working as a pseudo parameter layer and called input layer. Thus we have used a 5 layered network which includes 3 hidden layers. Specification of each layer is as follows:

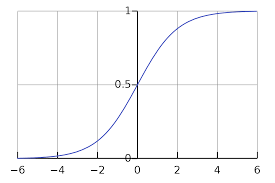
|  |  |  |  |
| --- | --- | --- | --- |
| Layer name | Number of neurons | Kernel initializers | Activation function |
| Input Layer | 128 | normal | Sigmoid |
| Hidden layer 1 | 256 | normal | ReLU |
| Hidden layer 2 | 256 | normal | ReLU |
| Hidden layer 3 | 256 | normal | ReLU |
| Output Layer | 1 | N/A | Linear |

Table 4.1.1: Layer specifications

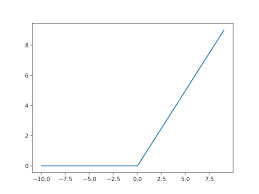
**Kernel Initializers**: The initial weights for the connection of a neural network set according to some method, there are some predefined kernel initializers in our Sequential python module. Normal initializer is the most used method; it generates a normally distributed weights matrix.

**4.2 Activation functions:**

As indicated by name these functions determine the state of neurons, if a neuron is on or off. If a neuron is in “On” state then only it can send signal to the next layer, i.e. activation function acts like logical gates in neural network. Activation function also generates the output of a neuron. Activation functions in our neural networks are both inspired from characteristics of activation function and experimental trials.

 4.2.1 Sigmoid activation function: This function works on standard sigmoid function, and produces results from 0 to 1. For generating the input layer from a very few parameters to study a regression problem, sigmoid is supposed to work best as it prevents jumps in output values. We have experimented with linear, ReLU and tanh activations in input layer, but no significant changes are seen from changing activation function in input layer. At higher values of x sigmoid results in values very close to 1(*Vanishing gradient problem*), which makes differentiating output values at high input value, but as stated experimental trials are also taken into account and sigmoid works well in our problem.

R(x) =

4.2.2 Rectified Linear Unit (ReLU): Despite its linear appearance ReLU is not a linear function and provides the same benefit as sigmoid, also it is extremely useful for regression problems. This is only to be used in hidden layer as it tends to create some dead neurons, as if a neurons emits no signal to the next layer and has no effect in output, in further iterations weights of the incoming connection to the particular neuron remains unchanged and neurons stays dead for any further iteration. As we have only two input variables at the start and we have already exploded them to generate 128 input layer parameters, we expected to afford some neuron loss in our hidden layers and our expectation is met from the experiments. ReLU has very simple formula and computationally less expensive then sigmoid and tanh.

4.2.3 Linear activation function: It is just a linear function, most suitable to use for generating numerical output, like in our problem. Used in our output layer to give the crack growth rate.

4.3 Loss function: Each machine learning algorithm requires a loss function to compare and evaluate its output with target values, and change the algorithm parameters accordingly. Our neural network’s weight matrices are also updated after each iteration, so a loss function is to be selected to guide our dumb neural network in getting smart and efficient. As we have a numeric output and a numeric target value, mean squared error (MSE) is an obvious choice.

1. **Using crack growth prediction to determine fatigue life**

Once we have the model to predict the crack growth rate corresponding to the stress intensity factor range, we can easily calculate the fatigue life with easy mathematical formulation. We assume that we are give a initial crack length (*l0*) and a critical crack length (*lc*). With each cycle crack length grows by a value (*dl)*, which we can predict with our neural network model at the current stress intensity factor range and stress ratio.

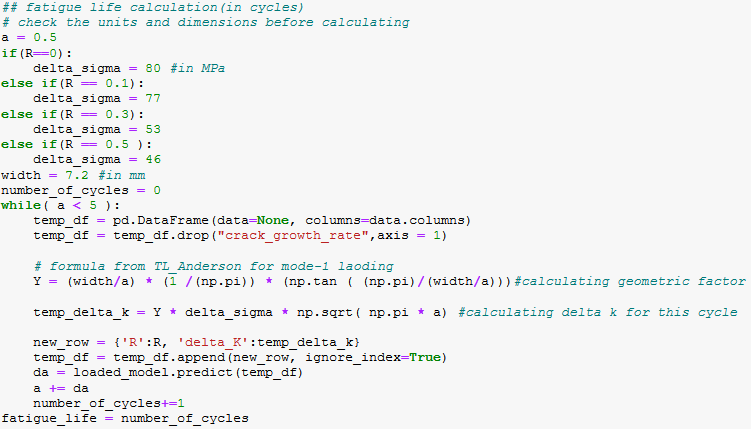
∆K can be calculated as:

Y= geometrical factor

g is a “specimen type” dependent function

W= width of the specimen

After each cycle new crack length will be increased by *dl* amount, we simply have to run this loop until our specimen reaches the critical length (*lc*) and keep the cycle count. A code snip this calculation is attached here(numerical values are take according to our specimen specifications):



Our data is for a specimen under mode-1 loading, so geometric factor function is chosen from The Fracture mechanics by TL-Anderson book [3].

1. **Results so far(till Mid-term evaluation)**

Our trained neural network has performed excellently on our mined dataset from various perspectives. We compared the predicted crack growth rate results with the experimental

Data and a comparison between crack growth rates calculated from classical Paris law method and our predicted crack growth rate has also been done and at low ∆K values the neural network giving better result. We have also tested the neural network algorithm on a stress ratio(R) value that it has never seen before, and as we know the parameters in Paris law are dependent on stress ratio. In Paris law we need the data for the particular stress ratio values to formulate our model and we can only calculate da/dN for that stress ratio. On the other hand, in this neural network algorithm we can predict the crack growth rate on any R value, irrelevant to the availability of data on that particular R. For better understanding results are shown in graphical form:

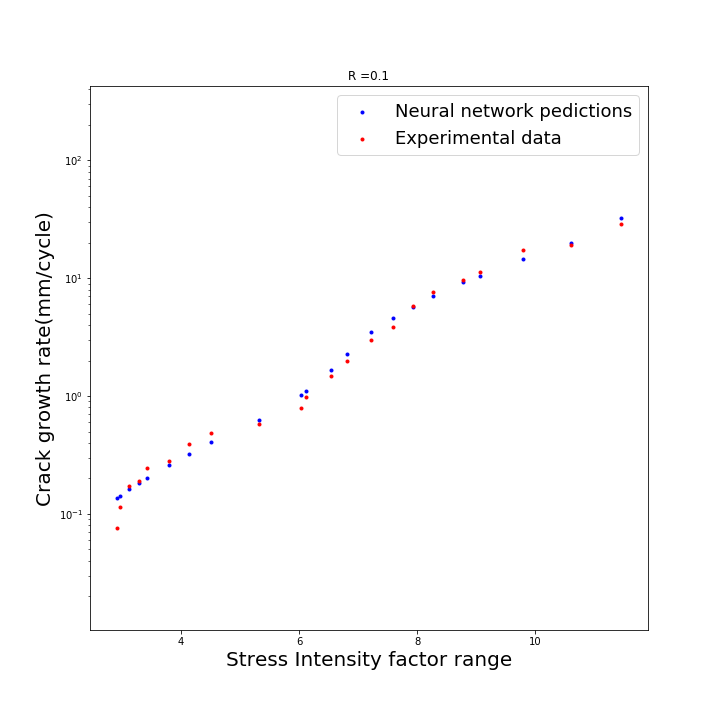


Figure 5.1: Neural network performance on an unseen stress ratio(R)

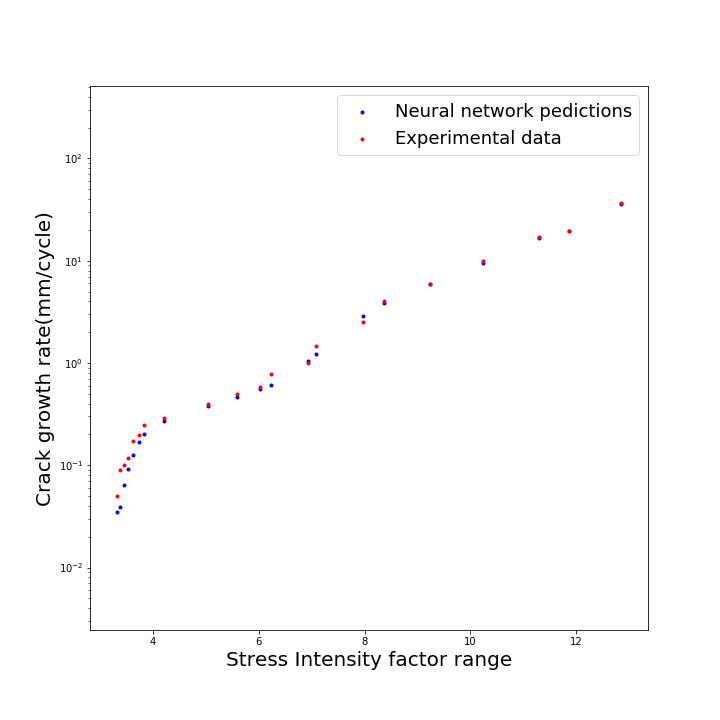


Figure 5.2: Neural network performance with experimental data.

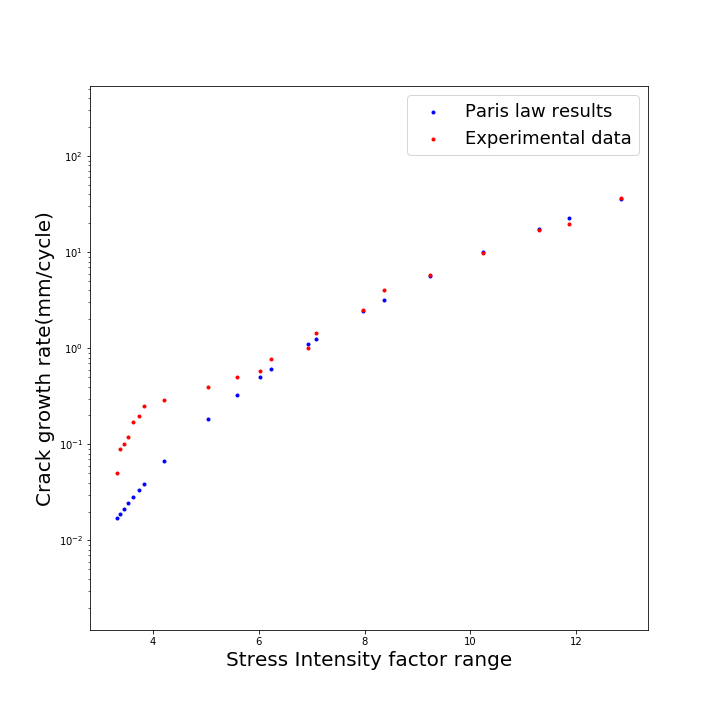


Figure 5.3: Paris law calculated values with experimental data.

1. Work Remains due to COVID-19 Pandemic

After mid-term the algorithm was to be tested on our own dataset generated from laboratory experiment to show the practical applicability of this neural network algorithm and validate this algorithm. According to the newly generated dataset, new modification and improvements were to be done in the neural network. Our neural network failed to achieve comparative performan-ce to the Paris law at high stress intensity factor range(∆K) values, so the improvement for the same had to be done.

**Submitted by: Submitted to:**

Vinod Khoja Prof. VIDIT GAUR

B.Tech (IV), MIED Assistant professor, MIED

Enroll. No. 16117098

**References and links:**

1. **Wang, Hongxun & Zhang, Weifang & Sun, Fuqiang. (2017). A Comparison Study of Machine Learning Based Algorithms for Fatigue Crack Growth Calculation. Materials. 10. 10.3390/ma10050543.**
2. **Anderson, T. (2017). Fracture Mechanics. Boca Raton: CRC Press,** [**https://doi.org/10.1201/9781315370293**](https://doi.org/10.1201/9781315370293)