**Introduction:**

Aircrafts operating in dusty environments suffer from structural damage to the gas turbine engine (GTE) components due to the ingestion of sand and other particulate matter. This leads to higher operational costs and in some tragic cases, loss of life. Particulate separators at the intake keep out particles larger than 80 µm in diameter from entering the hot-section of the GTE. Smaller particles which pass through, melt in the combustor and cause structural damage to the thermal barrier coatings (TBCs) which offer protection against high thermal loads on the gas turbine blades. The molten sand particles, which are a mixture of Calcium, Magnesium, Aluminum, Silica oxides along with trace amounts of other compounds and are referred to as CMAS. The molten CMAS deposits, chemically reacts and eventually infiltrates the TBC leading to altering the thermal properties of the TBC and thus reducing its life[1][2].

To develop mitigation strategies against CMAS attack, it is important to investigate the dynamic wetting of the molten CMAS on a surface coated with TBC. A surface with higher wettability has a larger surface area of contact with the molten droplet increasing its chance of failure. A volume of fluid (VOF) framework, commonly employed to mimic the flow of immiscible fluids, is a useful numerical tool to study surface wettability. In a VOF method, one way to compute the surface tension at the interface is done by using the continuum surface force (CSF) model. The resulting surface tension force is given as where, is the surface tension, is the interface curvature and is the interface normal vector. An accurate computation of is essential for interface reconstruction. One of the techniques used to compute curvature is the height function (HF) approach which uses the volume fraction from a local stencil. Following the work of Patel et al. [4] and Qi et al. [6], a neural-network based surrogate model is developed to compute the curvature as a function of local volume fraction.

**Neural Network Architecture:**

Following [6], a deep neural network is used to develop a function which takes in the volume fraction in the vicinity of the interface as an input and computes the corresponding curvature. The great success of deep neural networks in recent years is due to its ability to generalize large amounts of data at a much faster rate. This is achieved through a large number of model parameters thorough multiple neurons and multiple layers and clever use of graph methods to computes gradients. In this work, a feedforward neural network or a multi-layer perceptron (MLP) is designed using PyTorch [10] with three distinct layers: Input, Hidden and Output layer.

The network architecture in this work comprises of 9 inputs neurons, 2 hidden layers with 100 neurons, and 1 output layer as shown in Figure 1.

Diagram

Description automatically generated

Figure 1. Architecture of the neural network [3].

The data from the input layer is passed through a linear model (nn.linear) with random weights and biases. The linear transformation for a single layer is given

where is the input data vector, is the matrix with all the weights and is the bias vector. The key point in random initialization is to break the symmetry because if all the weights are assigned to zero then all the hidden layer neurons will be doing the same exact calculation. These weights and biases are optimized as we train our model by using an iterative approach known as backpropagation which the minimizes the cost function and improves the weights based on the result. The biases represent how far off the predictions from the actual value, and they make up the difference between the functions output and the intended inputs. In contrast, weights can be defined as the strength of the connections. In other terms, weights impact the amount of influence a change in the input will have upon the output.

The linearly transformed data is then passed through a non-linear activation function before being passed onto the hidden-layers.. The responsibility of the activation function is to check if an incoming value is greater than a threshold or a critical number. Also, this function outputs a small value for small inputs, and larger values for values that surpass a threshold. Activation functions are used because they add non-linearities to the neural network which allows the network to learn operations and in the absence it the model will simply be a set of linearly dependent vectors. The activation function used in this work is referred to as Rectified Linear Unit (ReLU), which returns 0 if it receives a negative input, and if receives a positive value, it will return the same positive value. The ReLU activation function can be represented as

and a plot of it is shown in Figure 2. In practice, the ReLU activation function has been observed to perform faster compared to sigmoid of tanh functions.

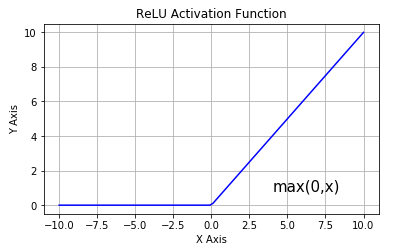


Figure 2. Plot of the ReLU activation function[7].

Backpropagation algorithm is at the heart of machine learning and deep-learning algorithms as it enables extremely fast gradient computations with respect to weights and biases to minimize the cost function. The goal of training a neural network is to optimize these weights and biases to minimize the cost function which is the mean square error in our case. The mean square error is defined as

MSE =

where yi is the output, ti is targets (corresponding curvatures) and N is the size of our dataset.

Adam is an optimization algorithm used to update network weights iteratively based on the training data. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. This optimizer combines the advantages of two other extensions of stochastic gradient descent which are Adaptive Gradient Algorithm and Root Mean Square Propagation. AdaGrad uses a per-parameter learning rate which improves the performance on a problem with sparse gradients.

**Data Generation:**

Following [6], circles with different radii were generated to train and develop the model. A total of 65 circles were generated between a radii of 0.0024 and 0.46. Basilisk [11] simulation software was used for generate the circles on a uniform mesh of size 1x1 with 2048 cells along each direction. For each cell containing the interface, the corresponding volume fractions from nine neighboring cells (Cij) were extracted. This is illustrated in Figure 3.

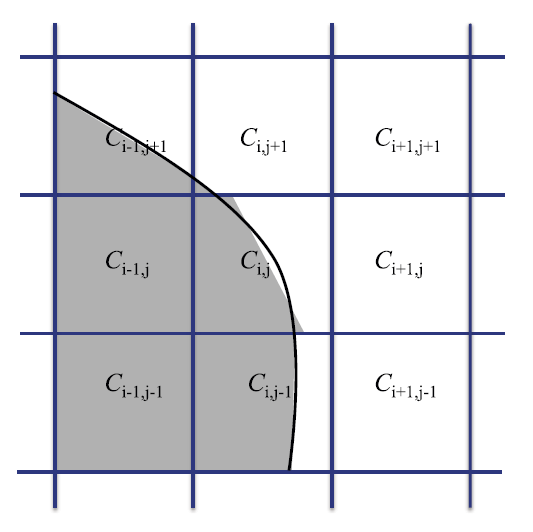


Figure 3, This figure shows an interface that intersects several computational cells. Picture reference: Qi et al [6]

The data generation was carried out on the DeepThought2 cluster at the University of Maryland. The dataset was randomly divided into 3 parts with the distribution of 70%, 10% and 20% for training, validation, and testing, respectively. More specifically, 585,180 data inputs for training, 79,911 for validation and 145,296 for testing. The input to the network is a 1x9 column data which represent the volume fractions, and the output is 1 column representing the scaled curvature. An user-defined data reader was written as a Python class to read the volume fraction dataset. Then, in the main function learning rate and batch size were defined and, and a loop to cycle through the data and train the model with the specified batch sizes was implemented. The learning rate is used in the stochastic gradient based optimizer and can be defined as a parameter that controls the amount of change in the model in response to the estimated error each time the model`s weights are revised. Generally, learning rate is a positive number between 0.0 and 1.0. To put it in another way, learning rate regulates how fast the model is adapted to the problem. Batch size refers to the sample size used in an iteration where epoch, is the number of passes of the entire training dataset that has been completed by the network. In the current network, the batch size is 256, the learning rate 0.001 and the training/validation was carried over 200 epochs.

After the training process was completed, the validation was conducted simultaneously along with training and the stopping criterion was determined when the training and the validation loss stopped improving. Also, this process can help us with adjusting our hyperparameters. At the end of every training epoch, the average loss for the validation process was calculated. Using this value, we can control where to stop the training process for our model. The code implementation for validation process was similar to training process. Another reason that validation data set was used is to analyze if overfitting or underfitting has happened to our model. Overfitting is when our model can classify data in the training data set but is not able to make accurate classifications on the data it was not trained on. In contrast, underfitting is defined as when a network cannot classify the data in which it was trained on. Finally, testing data was used to ascertain the efficacy of the network on the ‘unseen’ data. The testing loop iterates over the data and calculates the average loss of over the entire testing data.

**Results:**

After the execution was done for the training part, loss per epoch was found. A plot of the loss-per-epoch is shown in Figure 6. The average testing error at the 200th epoch was found to be 1.17283E-5, which is comparable to the values reported in [4][6].

Figure 6, Validation and Training loss bs Epoch graph

By looking at the graph we can observe that both validation and training losses are reducing which is a positive sign because it shows our model is becoming more accurate in terms of producing the correct output. Moreover, we can detect if the model has over or underfitting. To illustrate, if we had an overfit model, the training and the validating loss curves would have been far from each other, validation loss would have decreased gradually upon adding training samples, or the training loss was unreasonably low which very marginally increased with adding training samples. As opposed to an underfit data which might have an increasing training loss upon adding training samples or training and validation loss being close to each other at the end of the graph.

**Conclusions:**

In this work, a neural network was successfully implemented, trained, validated and tested to estimate the interface curvature as a function of the volume fraction from the neighboring cells. The preliminary results indicate a good match with results from the published literature [4][6] but additional work is necessary to fully develop the model and evaluate its performance. This includes

1. Testing the sensitivity of the model to the hyperparameters of the network such as number neurons, learning rate, batch size etc.
2. Test sensitivity to the sampling of training and validation data which is referred to as cross validation.
3. Test the model against different interface shapes (sine waves, ellipses etc.).
4. Implement the neural network model in Basilisk and evaluate performance.

**References:**

1. Padture, Nitin P., Maurice Gell, and Eric H. Jordan. "Thermal barrier coatings for gas-turbine engine applications." *Science*296.5566 (2002): 280-284.
2. Nieto, Andy, et al. "Calcia–magnesia–alumina–silicate (CMAS) attack mechanisms and roadmap towards Sandphobic thermal and environmental barrier coatings." *International Materials Reviews* (2020): 1-42.
3. IBM Cloud Education. (n.d.). What are neural networks? IBM. <https://www.ibm.com/cloud/learn/neural-networks>.
4. Luis Jalabert October 28, Jalabert, L., 28, A. O., Andy, 22, N. N., Neda, &amp; \*, N. (2017, October 26). A PyTorch tutorial – deep learning in Python. Adventures in Machine Learning. <https://adventuresinmachinelearning.com/pytorch-tutorial-deep-learning/>.
5. Ognjanovski, Gavril. “Everything You Need to Know about Neural Networks and Backpropagation - Machine Learning Made Easy...” Medium, Towards Data Science, 7 June 2020, towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a.
6. Patel, H. V., Kuipers, J. A. M., & Peters, E. A. J. F. (2018). Computing interface curvature from volume fractions: a hybrid approach. Computers & Fluids, 161, 74-88. <https://doi.org/10.1016/j.compfluid.2017.11.011>
7. PyTorch documentation. PyTorch documentation - PyTorch 1.9.0 documentation. (n.d.). <https://pytorch.org/docs/stable/index.html>.
8. Qi, Yinghe & Lu, Jiacai & Scardovelli, Ruben & Zaleski, Stéphane & Tryggvason, Gretar. (2018). Computing curvature for volume of fluid methods using machine learning. Journal of Computational Physics. 377. 10.1016/j.jcp.2018.10.037.
9. Sharma, Aditya. “Understanding Activation Functions in Deep Learning: Learnopencv Staging.” *LearnOpenCV Staging | OpenCV, PyTorch, Keras, Tensorflow Examples and Tutorials*, 4 May 2021, learnopencv.com/understanding-activation-functions-in-deep-learning/.
10. Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019): 8026-8037.
11. http://basilisk.fr