ANALYSIS OF TIME SERIES FORECASTING IN APPLICATION TO SOLAR ENERGY HARVEST

By

FADHIL ALI ALSAHLANEE

Bachelor of Science in Physics

Oklahoma State University

Stillwater, Oklahoma

2017

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2022

ANALYSIS OF TIME SERIES FORECASTING IN

APPLICATION TO SOLAR ENERGY HARVEST

Thesis Approved:

Dr. Douglas R Heisterkamp

Thesis Adviser

Dr. Christopher Crick

Dr. H. K. Dai

ACKNOWLEDGEMENTS

First and foremost, I would like to thank our mighty God for the blessings awarded me.

To my family, I cannot thank you enough for all the great support you have given me who empowered me in this study program I was doing.

To Prof. Heisterkamp, thank you for your guidance and patience. Without you, I would not have been able to complete and understand this work. Also, special thanks to Profs. Crick and Dai for serving on my thesis committee.

Name: FADHIL ALI ALSAHLANEE

Date of Degree: MAY, 2022

Title of Study: ANALYSIS OF TIME SERIES FORECASTING IN APPLICATION TO SOLAR ENERGY HARVEST

Major Field: COMPUTER SCIENCE

Abstract: The promised future applications in solar energy harvest have been remarkably recognized. However, the hourly forecasting of normal solar irradiance (NSI) outputs is considered a problem due to the dynamic nature of meteorological information not only in a day but also across days. The thesis proposed three neural network models including a dense layer without a hidden layer (DNN_h0), a dense neural network with two hidden layers (DNN_h2), a dense neural network with two hidden layers associated with one intermediate metrological feature (air temperature: T) (DNN_h2T), and dense neural network with two hidden layers associated with 7 intermediate metrological features (DNN_h2F). These models would be used to forecast an hourly prediction of normal solar irradiance (NSI) across an entire day. As well as, we proposed two configurations to represent our datasets: FTC (sine-cosine) and 1H (one-hot) encodings. In addition, we used metrological features such as air temperature T and others to determine the effectiveness of a model's performance in terms of mean absolute error (MAE). We conducted two groups of experiments: single-step and multi-step prediction models by using one real-world dataset (NREL). As a result, the comparison is revealed that the (NSI) has an acceptable model performance in both FTC and 1H encodings for the multi-step models by using an intermediate metrological feature: air temperature T in the (DNN_h2T) model. Whereas the single-step model (DNN_h0) has shown slightly acceptance to find a well performance to predict the (NSI), while the (DNN_h2) model shows a significant (MAE) values in both encodings.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Forecasting is an important problem that spans many fields including energy, industry, government, economics, environmental sciences, and others [1]. A sequence of observations generates time series [2]. Nowadays, the promised future applications in solar energy harvesting have been remarkable recognized since the solar energy reaches the earth exceeds by far humankind's needs. The basic concept of collecting solar power is relatively simple, as a concentrate energy from the sun's rays to heat a receiver to high temperatures and inverted into electricity [3]. The prospective analysis of data related to solar energy can be exposed by considering real experimental data as a time-series taken from disciplines [4]. The primary objective of the time-series analysis is to develop mathematical models that provide plausible descriptions for sampling the datasets. A time series are taken one variable that is the time, while many algorithms can be used to achieve the predicted future results of the sensible. Solar harvest has shown considerable interests in recent decades were unsteady due to massive inter-day and intra-day fluctuation, seasonal effects, dirt, and hardware aging [14] [15] [16]. The environment has a notable impact, e.g. shades of buildings and trees reduce the harvest considerably [8]. Numerous algorithms have been devised that adjust the task schedule of sensor nodes online to achieve energy-neutral harvesting operations, and save as much energy as possible to aim at maximizing the energy-neutral node consumption. The high harvesting rates are related to predict the future of a long-term predictions at least one day [17] [18] [19] [20].

Objectives of the thesis

This work analyzes the time series of one meteorological dataset applied in solar energy harvest. The hourly predictions of solar irradiance are considered the main problem due to the dynamic nature of radiant energy and meteorological data across days [10]. The air temperature is also one of the weather dependents uncontrollable that conjoined to the existence of clouds and can be fed into the solar irradiance predictions [22]. The thesis is an evaluation study of hourly time series forecasting models to predict the direct normal solar irradiance (NSI) associate with an intermediate meteorological feature: the air temperature (T) by using three models categorized as artificial intelligence models (artificial neural networks), then used a comparative approach among the models in terms of their performances. The proposed models including one dense layer (i.e. without hidden layer), dense neural network (two dense hidden layers), and dense neural network (two dense hidden layers) associated with intermediate feature: air temperature (T). However, these models would use to make the predictions for one dataset over the 9 years of the time period 2011 through 2020. Fourier analysis (sine-cosine) and one-hot encodings are suggested to compare the effectiveness of these models performance. Thesis forecasting work experiments are divided into two groups:

1. Single-step prediction models: one hour into the future considering the daytime hours for a single meteorological feature value such as (NSI).
2. Multi-step prediction models: each model needs to train and predict a range of features values. Unlike a single-prediction, where only a single future point is predicted, a multi-step forecasts a sequence of the intermediate meteorological features such as (T) to make the daytime hourly forecast and would be fed into the (NSI) prediction.

The results are discussed in terms of each model's performance. Comparing the performance among prediction models would be determining if our experiments would aid improve the prediction’s performance for the direct normal solar irradiance (NSI) values. Also, our work is an attempt to validating the assumption of using the both Fourier transformation and One-hot encodings on each model's performance that may impact forecasting results of the time series, and thus can be extending their usage in the literature.

CHAPTER II

REVIEW OF LITERATURE

In 2010, Jun Lu et al. [5] and 2012, Hejase, H. A. N. & Assi, A. H. [6] published a study of two models independently, which have been adopted the system performance for energy harvesting to predict the mean daily global solar radiation. In 2012, Cammarano, A. et al. [7] and 2013, Renner, C. [8] published independently a novel energy prediction models, which was able to leverage past energy observations to provide accurate estimations of future energy availability, and proposed and evaluated two methods that combine local information of a node's harvest pattern with global cloud cover forecasts. In 2015, Prema, V. & Rao, K. U. [9] and 2018, Sharma, A. & Kakkar, A. [10] they've independently proposed models for short-term prediction of solar irradiance from which solar power can be predicted, adding to, series of experimental evaluations presented in terms of forecast accuracy, correlation coefficient and root mean square error (RMSE). In 2019, Ismail Fawaz, H. et al. [11] have reviewed the deep Learning techniques and some applications on Time-Series analysis were used and resulted good, as well as, a novel model of boosting ensemble strategy to demand forecasting systems by implementing a novel decision integration model. As well as Kilimci, Z. H. et al. [12] published an improved demand forecasting model using a deep learning approach and a proposed decision Integration strategy for the supply chain.

In 2019, Donghun Lee & Kwanho Kim [22] they suggested three PV power output prediction methods such as artificial neural network (ANN), deep neural network (DNN), and long and short term memory (LSTM) based models that are capable to understand the hidden relationships between meteorological information and actual PV power outputs. Also, they proposed LSTM based model is designed to capture both hourly patterns in a day and seasonal patterns across days, and found out that the proposed ANN-based model fails to yield satisfactory results, and the proposed LSTM based model successfully better performs more than 50% compared to the conventional statistical models in terms of mean absolute error as shown in figure 2.1.

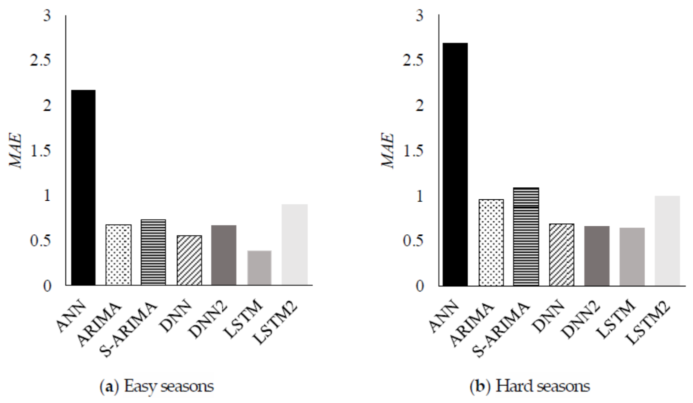


Figure 2.1 shows the performance comparisons according to easy and hard seasons [22]

Donghun Lee & Kwanho Kim's research has opened up an entire new gap in our understanding of neural networks used in predictions. Numerous efforts have been proposed in the literature. Thus, this knowledge remains limited in such fields. This work explores the unknown field and contributes to minimizing this gap.

Although statistical time series forecasting methods are utilized in the literature, there are a limited number of studies that utilize deep artificial neural networks, however, the effect of using multivariate data on solar radiation forecasting using a deep learning approach which proposed a multivariate forecast model that uses a combination of different meteorological variables. In 2020, Sorkun, M. C. et al. [33] proposed a multivariate forecast model that uses a combination of different meteorological variables, such as temperature, humidity, and nebulosity by using recurrent neural network (RNN) variation, namely a long short-term memory (LSTM), which they observed that temperature and nebulosity are the most effective parameters for predicting future solar irradiance. As well as in 2020, Nair, V. et al. [34] have leveraging the Fast Fourier Transformation, reduces the image convolution costs involved in the Convolutional Neural Networks (CNNs) and thus reduces the overall computational costs, which proposed a model that identifies the object information from the images. In addition, the idea that neural networks learn similar features on the same input inspired in 2020, Tancik, M. et al. [35] to publish a study of using a Fourier feature mapping to transform the effective the neural tangent kernel NTK into a stationary kernel with a tunable bandwidth. They have suggested an approach for selecting problem-specific Fourier features who greatly improves the performance of multilayer perceptron MLPs for low-dimensional regression. In 2020, Chitsaz, K. et al. [37] published a study to accelerate of Convolutional Neural Network using FFT-based split convolutions. Also, Wolter, M. et al. [41] proposed to combine Fourier methods and recurrent neural network architectures, and used the short-time Fourier transform to efficiently process multiple samples at a time.

As mentioned in the thesis objective, the thesis inspired part of the works in [22], and [34] by using the procedure of single-layer and multi-layer neural networks and evaluates proposed models such as linear, DNN1, and DNN2 to conduct experiments on the available meteorological features and predict both (NSI) and intermediate features (T). Unlike the [22] work, our dataset has hourly patterns of the meteorological features that were taken across daytimes only, ignoring the week or the day of the month. Also ignore the location of a specific region. Likewise, a simple Fourier feature mapping enables a multilayer perceptron (MLP) to learn high-frequency functions in low-dimensional problem domains [35], we proposed the Fourier analysis and one-hot encodings to conduct the time series predictions and determine the effectiveness of each model performance towards the NSI predictions [36]. It reduces the costs involved in neural networks and thus reduces the overall computational costs [37]. The thesis work provides a well performance comparison among the models to explore an inexplicit field of the literature.

CHAPTER III

METHODOLOGY

This chapter presents the method of preprocessing a chosen dataset and the method to build our artificial neural network models (i.e. models architecture) that are used to find the time series analysis forecasts. As mentioned in our thesis's objective earlier, the thesis conducts two groups of experiments: single-step and multi-step prediction models to the normal solar irradiance (NSI) associated to the air temperature (T) of the next hour. By training three neural network models including a dense layer without hidden layer (DNN_h0), a dense neural network with two hidden layers (DNN_h2), a dense neural network with two hidden layers associated with one intermediate metrological feature (air temperature: T) (DNN_h2T), and dense neural network with two hidden layers associated with 7 intermediate metrological features (DNN_h2F) using a TensorFlow-Keras library (version 2.6.0). We proposed two encoding representations: FTC (sine-cosine) and 1H (one-hot) for our dataset, the models' performance is conducted in terms of the mean absolute errors (MAE).

3.1 Dataset

The thesis is used one real-world dataset. The dataset is collected by the National Renewable Energy Laboratory (NREL): Solar Radiation Research Laboratory, which records samples per hour for an entire day. The dataset has been taken from Jan 1, 2011, through Jan 1, 2020. The NREL dataset contains 9 meteorological features including the avg direct normal solar irradiance (NSI) [kW-hr/m^2], avg air temperature (T) [deg C], avg atmospheric pressure (P) [mBar], avg wind speed (W) [m/s], avg peak wind speed (PW) [m/s], avg precipitation (Prep) [mm], and avg relative humidity (RH) [%], zenith angle (Z) [degrees], and azimuth angle (A) [degrees].

NREL Dataset Source:

https://midcdmz.nrel.gov/apps/daily.pl?site=IRRSP&start=20070619&yr=2020&mo=9&dy=14

To show the seasonality evolution of our meteorological features over the selected years, the evolution of the original dataset features is illustrated in appendix A.

3.2 Dataset Configuration

The 9 meteorological features in an original dataset is transformed into two different representations: FTC(sine-cosine) and 1H (one-hot) configurations. For the 1H encoding, we used the 24 bits for representing the hour of the day at the beginning of each year. Thus, the feature vectors used for hours, date and angles as follows:

1. FTC configuration: the total size of our feature vectors is 15, where hours, date and angles are transformed to sine and cosine components. Whereas the 7 features (RH, W, PW, P, PR, NSI, and T) values are copied without modification.
2. 1H configuration: the total size of our feature vectors is 61, where angles are transformed to sine and cosine components. The hours, and date are represented into 24 for the hours of the day and 26 for the date. Whereas the 7 features (RH, W, PW, P, PR, NSI, and T) values are copied without modification.

However, the deterministic feature values of hours, date and angles (i.e. the zenith and azimuth angles) would not need to be predicted as they can be calculated directly. The used codes are described in appendix B.

3.3 Dataset Splitting

Evaluating a model always boils down to splitting the available dataset into two sets: training, and validation [46]. We partitioned the data into three data sets as in the following:

Training dataset (2011 - 2017) (six-years), Validation dataset (2018) (one-year), and Testing dataset (2019) (one-year).

The split of the data ensures dividing it into the windows of consecutive readings and transformed our data into samples of inputs of past observations and outputs of future observations. The six years training set is used to fit the learning model, the one-year validation set is used for tuning the model's hyper-parameters, where to regularize the signs of overfitting/underfitting and calculating the loss function in terms of mean square error (MSE) to stop the learning process, whereas the one-year test set is used for evaluating the model's performance (i.e. the generalization errors) in terms of mean absolute error (MAE).

3.4 Dataset Cleansing and Normalization

In order to eliminate the noise in the seasonality evolution of the original meteorological features inputs as shown in appendix A. We found that the original data have sensor readings of -99999, which only happened 5 times at 2:00 AM hour, and in the first two years. Since that values cannot be valid for the prediction process, it must be a signal for a missing measurement. Also, it will not affect our predictions between 8:00 AM through 6:00 PM. However, without removing them will throw off the data pre-processing. We detected Outliers and replace them with linear interpolation of adjacent values. In addition, we used the options to toggle removing Outliers and to toggle among Min-Max scale, zero mean, and unit variance scaling. After replacing the missing measurement values, Outlier's detection found a lot more outliers and most of them were adjacent so we cannot have applied linear interpolation and it just clips the outlier the sigma equals to 3 value. Thus, the values are conducted to clean up the outliers of our dataset. So any distance value with an integer value greater than 3 are detected as outliers [29] [42] [45] [46] [47]. The details coding is described in the appendix C.

3.5 Sliding Window and Training Models

We used a sliding input window over a sequential sequence for the 7 numerical input feature vectors. At a time, t, all of the feature vectors from time t-w to t are concatenated into an input vector for a model. A model then makes a prediction for time t+k. Our experiments have used a setting: w = 24, k = 1, where w is the width (number of time steps: hours) of the input window, k is the predicted value. Thus, our models predicted all of the components of the input vector at time t+k, except for desired prediction item. To wrap up, the final dimensions of feature vectors conducted are 15 feature vectors in FTC encoding and 61 feature vectors in 1H encoding. We used the validation set to evaluate the models in terms of mean square error (MSE) to tune our models. Once the models are ready for the peak time, we conducted a test them within a test set to calculate the models performance. We used 250 epochs, and 128 batch size in training the models. Thus, the amount of errors in the predicted outputs in terms of mean absolute error (MAE) which can be calculated during each epoch. As the epochs go by, the model learns and its loss error (MSE) on the training set naturally goes down, and so does its prediction error on the validation set. However, after the epoch is finished, the validation error stops decreasing and actually starts to go back up. This indicates that the model has started to overfit the training data, while the loss function between training and validation sets is calculated [29]. Also, we used Adam optimizer which reaches the lowest value to track the loss faster and determine how much the weights are needed to be adjusted toward minimizing (MAE) values for the entire training dataset [22] [45]. The (MAE) values and loss function results are illustrated on the next chapter. The codes are described in appendix D.

3.6 Neural Network Models

Thesis models scripts, structures and summary are described in appendix E.

3.6.1 Baseline model

As mentioned previously for training our dataset, we built a simplest model by using the target value from time t to make prediction at time t+k ignoring earlier input values in the input window t-w to t-1. This model used to have a performance baseline as a point for comparison with the later models [45].

3.6.2 Dense without hidden layer (DNN_h0) model

We used a single dense layer forward fully connected neural network for the training dataset between inputs and outputs. The model is used all of the feature vectors in the input window, where there are no interactions between the predictions at each time-step and did not even place the most weight on the inputs to avoid one of the risks of random initialization. The set of our model as follows:

- input layer = (k features * t time samples of input window)
- output layer = (1 node), fully connect to input layer (used the sigmoid activation).

3.6.3 DNN (Dense Neural Network) models

We used stacks of 2 dense hidden layers forward fully connected neural network between the inputs and the outputs. Also, the model used 2 dropout layers between the first hidden layer and the second hidden layer, and between the second hidden layer and the output layer.

Our **single-step prediction model** called DNN_h2 (dense neural network with 2 hidden layers) that is used a set as the following:

- input layer = (k features * t time samples of input window)
- hidden layer 1 = (h1 nodes), fully connect to input layer (used ReLU activation)
- dropout layer1 (hidden layer 1)
- hidden layer 2 (h2 nodes), fully connect to dropout layer 1 (used ReLU activation)
- dropout layer2 (hidden layer 2)
- output layer (1 node), fully connect to dropout layer 2 (used Sigmoid activation)

The DNN_h2 model is trained using a loss function between the single output node that could be used to predict the NSI of the next time step. In our **multiple-step prediction model** called DNN_h2T (dense neural network with 2 hidden layers within two features: T and NSI) that would be used for predicting the next time step the two features (T to NSI) concatenated by using a Concatenate layer, where used a set for feeding the NSI from T and as the following:

- input layer = (k features * t time samples of input window)
- hidden layer 1 = (h1 nodes), fully connect to input layer (used ReLU activation)
- dropout layer1 (hidden layer 1)
- hidden layer 2 (h2 nodes), fully connect to dropout layer 1 (used ReLU activation)
- dropout layer2 (hidden layer 2)
- output layer 1 (1 node = T), fully connect to dropout layer 2 (used Sigmoid activation)
- Concatenate layer = fully connected to output layer 1 and dropout layer 2
- output layer 2 (1 node = NSI), fully connect to concatenate layer (used sigmoid activation)

The DNN_h2T model is trained using a loss function between the single output 2 nodes to predicted both the T and NSI of the next time step. However, one more **multiple-step prediction model** called DNN_h2F (dense neural network with 2 hidden layers within 7 features: RH, W, PW, P, PR, T, and NSI) that would be used for predicting the next time step the 7 features (RH, W, PW, P, PR, T, to NSI) concatenated by using a Concatenate layer, and it is trained using a loss function between the single output of 7 nodes, where used a set as the following:

- input layer = (k features * t time samples of input window)
- hidden layer 1 = (h1 nodes), fully connect to input layer (used ReLU activation)
- dropout layer1 (hidden layer 1)
- hidden layer 2 (h2 nodes), fully connect to dropout layer 1 (used ReLU activation)
- dropout layer2 (hidden layer 2)
- output layer 1 (1 node = RH), fully connect to dropout layer 2 (used Sigmoid activation)
- output layer 2 (1 node = W), fully connect to dropout layer 2 (used sigmoid activation)
- output layer 3 (1 node = PW), fully connect to dropout layer 2 (used Sigmoid activation)
- output layer 4 (1 node = P), fully connect to dropout layer 2 (used sigmoid activation)
- output layer 5 (1 node = PR), fully connect to dropout layer 2 (used Sigmoid activation)
- output layer 6 (1 node = T), fully connect to dropout layer 2 (used sigmoid activation)
- concatenate layer = fully connected to dropout layer 2 and output layers 1, 2,3,4,5, and 6.
- output layer 7 (1 node = NSI), fully connect to concatenate layer (used Sigmoid activation)

3.7 Experimentation

As mentioned earlier in the thesis's objective in chapter 1, the thesis has conducted two groups of experiments applied to a real-world dataset (NREL dataset) by leveraging the TensorFlow-Keras library (version 2.6.0). The first group of experiments is a single-step prediction model that would be forecasted the normal solar irradiance (NSI), by which including the models DNN_h0, and DNN_h2. These models are trained on a training set for a loss function on out of the next time step (i.e. one hour ahead). In the next step, the models are validated using a validation set, where an output node corresponds to our prediction's model for the NSI. However, the deterministic feature values of time, date and angles are not need to be forecasted as they can be calculated directly as mentioned earlier.

The second group of experiments is a multi-step prediction which are used models including DNN_h2T and DNN_h2F of the next step prediction (i.e. one hour ahead) that would be used to conducting the forecast of other intermediate features such as T. However, the DNN_h2T model is used to resulting the first output layer of prediction (T) to feed up our goal the (NSI), and the second output layer is used to resulting our desired (NSI) prediction (i.e. DNN_h2T have two output nodes). Likewise, the DNN_h2F is used to make a verification that it would perform about the same as our focused two features model (i.e. DNN_h2T). That is, the models are trained on a training set for a loss function between the predicted features and the desired outputs.

The difference between the second group models and the previous ones is that the links are added in series by using a Concatenate layer from all of the hidden layers to the related output nodes; however, including an intermediate feature (T) to feeding and corresponding to the target output (NSI). The hidden layers (h1 and h2) are set the number of nodes as the following: hidden layer 1 (h1) set to 128 nodes, and hidden layer 2 (h2) set to 32 nodes.

Also, we used two of the regularizations; however, the dropout layer within a rate value of (0.3) between the hidden layers (h1 and h2) and an output layer as mentioned in the DNN models (subsection 3.6.3), as well as assigning loss weights to take a value (0.5) for each output. Also L2 regularizer is used with a factor's default value (l =0.01) in both DNN_h2T and DNN_h2F models, that is, might be prevented our models from overfitting [48].

Both experiments (single-step and multi-step predictions) are conducted for both FTC and 1H encodings. The performance of each model are calculated in terms of mean absolute error (MAE). A comparison of the performance of the first group prediction models and the second group prediction models would be determining if the training makes explicit forecasts of the other features, where maybe helped in the improvement of the prediction to the targeted (NSI) value. The models structures are described in appendix E.

3.8 Models Performance

The thesis used the mean absolute error (MAE) based on the difference between the predicted values and the true values for the total number of data points used. The MAE was obtained by standardized values using Equation (3.1) [38]:

$$\mathrm{MAE} = \frac{\sum_{\mathrm{i}=1}^{\mathrm{n}} |\mathrm{Y_i} - \mathrm{X_i}|}{n} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots. (3.1)$$

Where n is the total number of data points used in the testing, $\mathrm{Y_i}$ are the predictions and $\mathrm{X_i}$ are the true values [39] [40].

CHAPTER IV

RESULTS

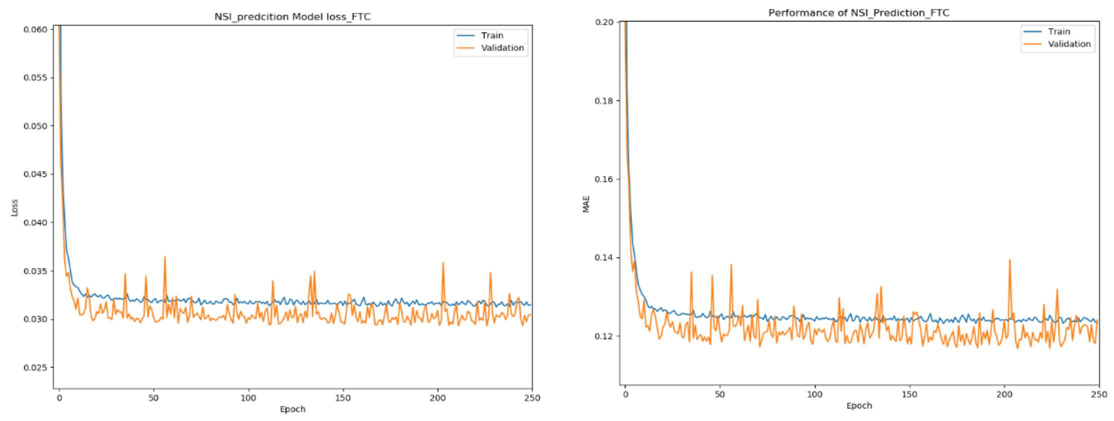
The results for the single-step and multi-step prediction models including DNN_h0, DNN_h2, DNN_h2T, and DNN_h2F as mentioned earlier in subsections 3.6.2 and 3.6.3 are presented and discussed by illustrating their performance in terms of mean absolute error (MAE). Also, we presented the models' performances in both proposed representations: FTC and 1H encodings, as mentioned earlier in section 3.2. The baseline model performance (MAE) has resulted as a point for the comparison to other models, by which the training set is given a value of (**0.1419**) and the validation set is given a value of (**0.1365**). We conducted base experiment and several additional experiments (investigations) to tackle both underfitting and overfitting problems.

4.1 Base experiment

The trained single-step models (DNN_h0, and DNN_h2) are shown different values of performance to forecasting the (NSI) that would be considered in terms of the mean absolute error (MAE). As mentioned in the previous chapter and through our base experiment, we determined the required settings of the hyper-parameters set as number of epochs = 250, batch size = 128 and a dropout layer rate value of (0.3) [22]. On the next pages, figures 4.1 and 4.2 are presented the experimental results that illustrated the changes of the NSI-loss function and the NSI-MAE values of our DNN_h0 model in both FTC and 1H encodings during the training session, respectively. Since the training set is used to train the model, whereas the validation set is used to estimate and tune the parameters of the model. That is, decided when to stop training and set meta-parameters of the models. The loss function and the MAE values of training and validation sets would be decreased as the number of epochs increased which have very convergent numbers (i.e. there is no big difference in values by taking its number as is during the training session). In which indicates that our simple DNN_h0 model successfully attempts to search for some well solutions against the given validation dataset by gaining errors less than the baseline model. However, figures 4.3 and 4.4 are illustrated the base experiment results of the DNN_h2 model in both encodings (FTC and 1H) respectively. The DNN_h2 model learning curves are shown a sign of underfitting in both encodings, except a slight difference in the loss function that tends to be alleviated the sign of underfitting in 1H encoding.

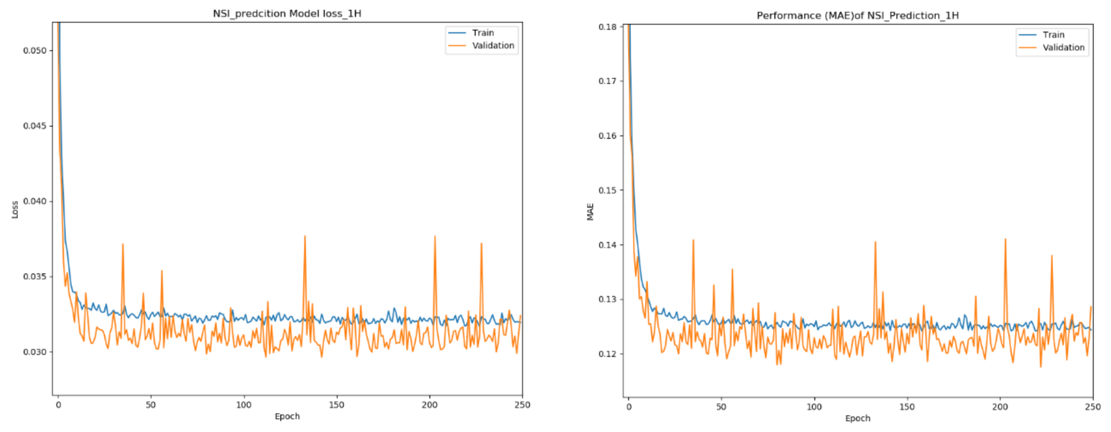
However, the learning curves and the NSI mean absolute error of the most complex model: DNN_h2T are shown much better of results compared to our baseline model values. As well as the 7 features (RH, W, PW, P, PR, NSI, and T) model: DNN_h2F, which is used to verify the performance as the two feature (NSI and T) model (i.e. DNN_h2T), that would not improve over the T and NSI. Thus, the MAE values of both NSI and T are illustrated a comparatively sign of underfittings which is a sign of lower errors in validation in the FTC encoding, whereas a sign of searching to find a well fitted in the 1H encoding in the loss function and MAE values. Figures 4.5, 4.6, 4.7 and 4.8 are illustrated the NSI learning curves of the DNN_h2T and DNN_h2F models in both encodings, respectively. Table (4.1) shows the MAE values of the base experiment. However, the DNN_h2, DNN_h2T models are still needed to be investigated for more experiments corresponding to the underfittings and overfittings problems, by which the underfitting means that the model has a poor relationship with the training set.

We conducted additional experiments (i.e. investigations) later in this chapter to alleviate these major challenges.



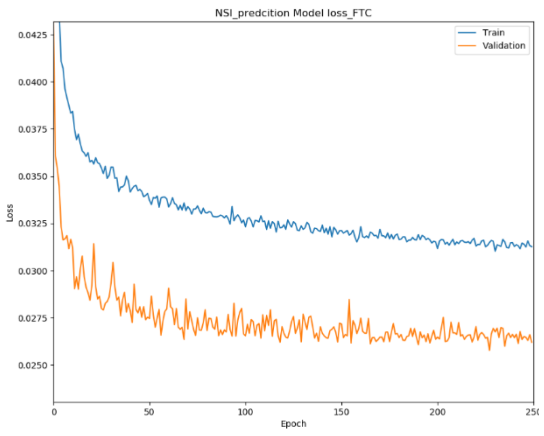
Loss function MAE

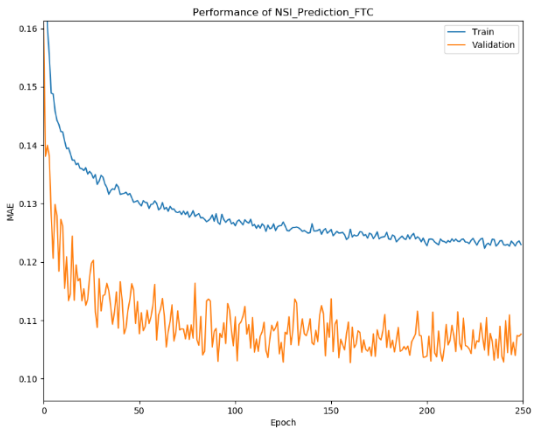
Figure 4.1 DNN_h0 model NSI- loss function and MAE learning curves in FTC encoding.



Loss function MAE

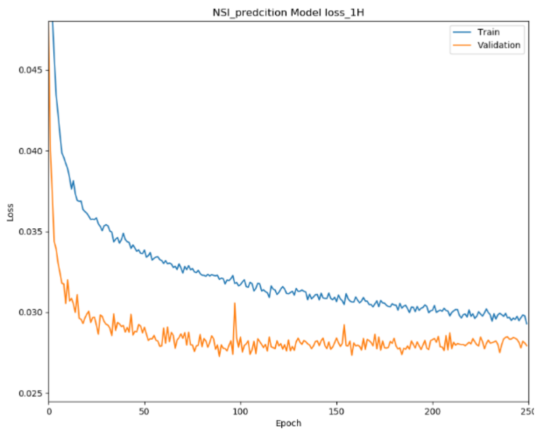
Figure 4.2 DNN_h0 model NSI - loss function and MAE learning curves in 1H encoding.

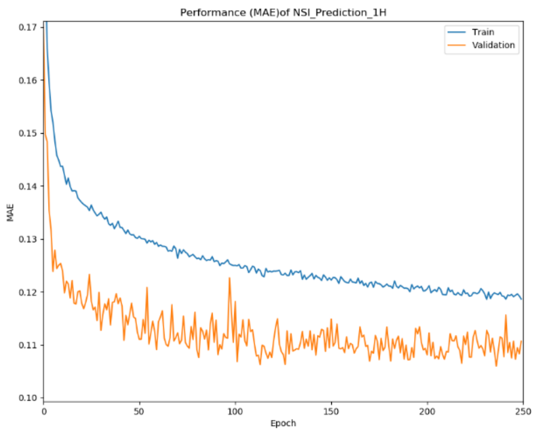




Loss function MAE

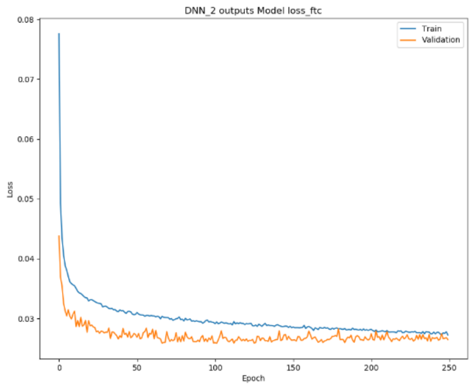
Figure 4.3 DNN_h2 model NSI- loss function and MAE learning curves in FTC encoding.

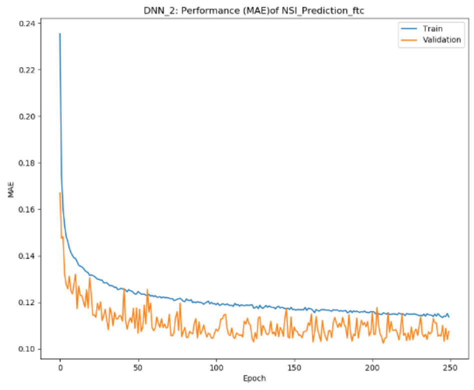




Loss function MAE

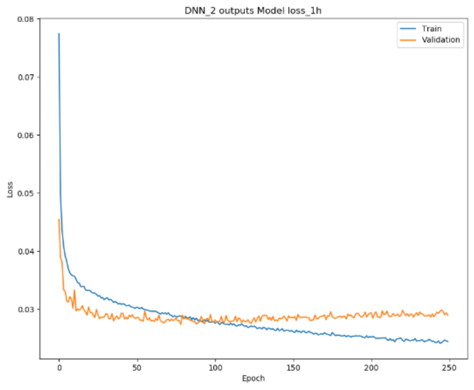
Figure 4.4 DNN_h2 model NSI- loss function and MAE learning curves in 1H encoding.

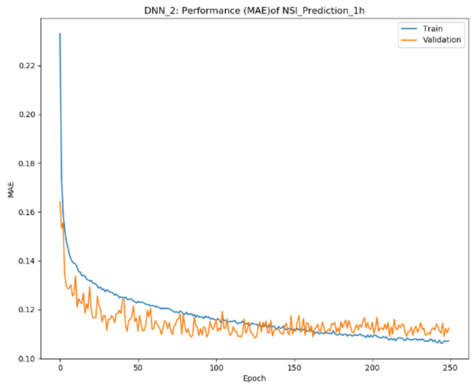




Loss function MAE

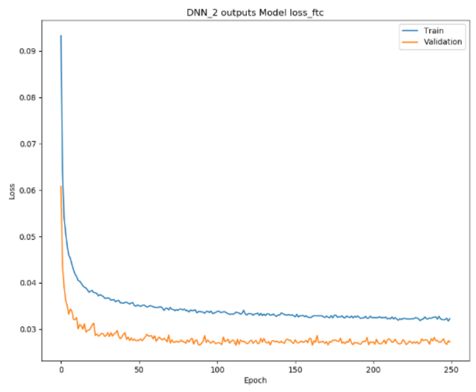
Figure 4.5 DNN_h2T model NSI- loss function and MAE learning curves in FTC encoding.

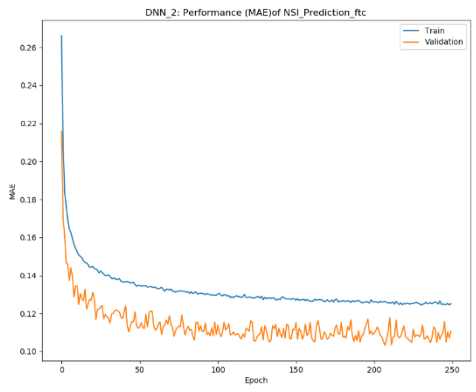




Loss function MAE

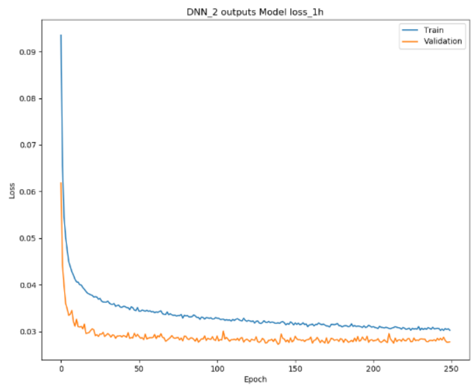
Figure 4.6 DNN_h2T model NSI- loss function and MAE learning curves in 1H encoding.

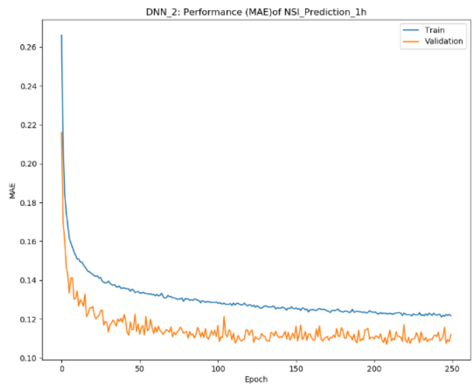




Loss function MAE

Figure 4.7 DNN_h2F model NSI- loss function and MAE learning curves in FTC encoding.





Loss function MAE

Figure 4.8 DNN_h2F model NSI- loss function and MAE learning curves in 1H encoding.

Table 4.1 shows the MAE results of the base experiment.

Models	**MAE**				
	base experiment				
	FTC encoding		1H encoding		
		Train	Val	Train	Val
Baseline	NSI	0.1419	0.1365	0.1419	0.1365
Single-step prediction: DNN_h0 and DNN_h2 models epochs = 250 , batch size = 128 , opt=Adam					
DNN_h0	NSI	0.1233	0.1240	0.1244	0.1286
DNN_h2	**h1 =128 ,h2 =32, Dropout (0.3)**				
	NSI	0.1230	0.1076	0.1186	0.1107
Multi-step prediction: DNN_h2T and DNN_h2F models epochs = 250 , batch size = 128 , opt=Adam					
DNN_h2T	NSI	0.1136	0.1074	0.1072	0.1123
	T	0.0373	0.0208	0.0389	0.0237
DNN_h2F	NSI	0.1251	0.1108	0.1216	0.1119

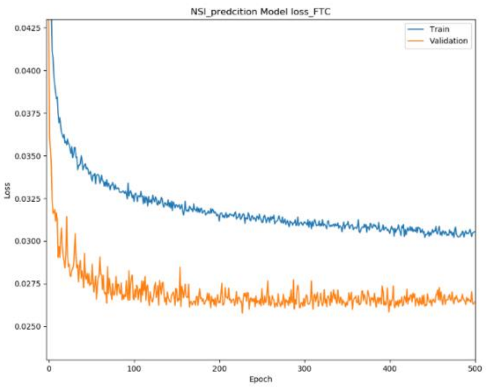
4.2 Investigation experiments

In order to tackle the underfitting and overfitting problems, we conducted several investigation experiments that may alleviate and reduce these problems in our DNN_h2 and DNN_h2T models.

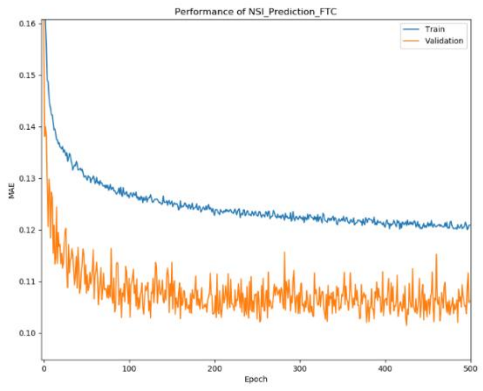
4.2.1 Change the Number of Epochs

That is, increasing the number of epochs (i.e. training time) by 500, 1000, 2000 and 20000 may minimize the underfitting issues while maximizing the generalization ability of our models in terms of the loss function values using the mean square error (MSE) and the mean absolute error (MAE). Also, we used the same base experiment settings of which the batch size = 128, number of nodes of the hidden layers (h1 = 128, h2 = 32) and dropout layers' rate value set to (0.3) in both encodings. The DNN_h2 model MAE values are shown slightly a sign of overcome the underfitting in the NSI- loss function learning curves during the 500 epochs in 1H encoding and still suffering the underfitting in FTC encoding, whereas the NSI-loss function values are tending to be overfitted after the epoch 400. For the number of epochs 1000, the NSI-MAE values of DNN_h2 model are shown very convergent fitted learning curves associated with 1H encoding and has still suffered the underfittings within the FTC same as the number of epochs goes 500 through 1000, whereas the NSI-loss function values have the same patterns as of 500 epochs in which tend to be overfitted after the epoch 400.

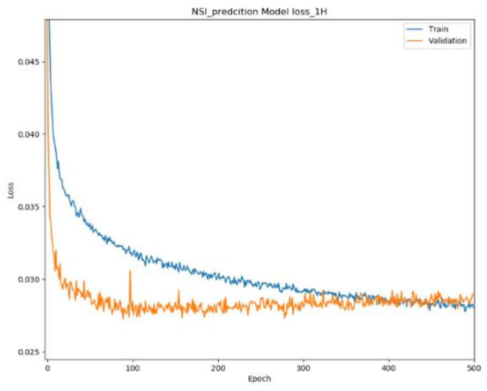
However, the DNN_h2T model trained for more training time on 500, 2000, and 20000 epochs that is resulted and shown more stabled validation curves and decreased training curves. Thus, the NSI training error is decreased and the NSI validation error stays about the same values, in which this behavior may expect after starts fitting well at epochs 500 in the FTC. Whereas the increase the number of epochs to 20000 would expect a sing of overfittings. On the following pages, the single-step model: DNN_h2 the NSI- learning curves on number of epochs 500, and 1000 are shown in figures 4.9 and 4.10 for both encodings respectively. Whereas the multi-step model: DNN_h2T the NSI learning curves on number of epochs 500, 2000, and 20000 are illustrated in figures 4.11 ,4.12, and 4.13 in both encodings respectively. Table (4.2) shows the MAE results for both models on the given number of epochs (500, 1000, 2000, and 20000).



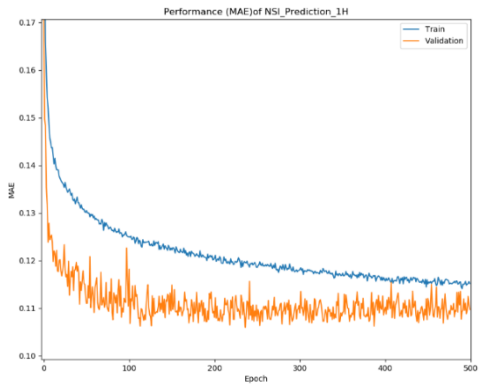
Loss function-FTC



MAE-FTC

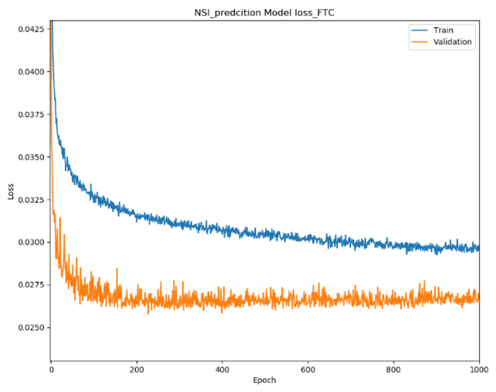


Loss function-1H

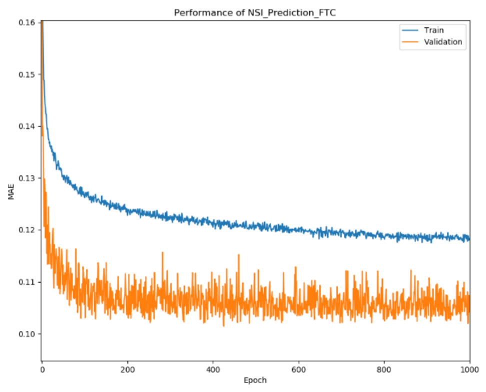


MAE-1H

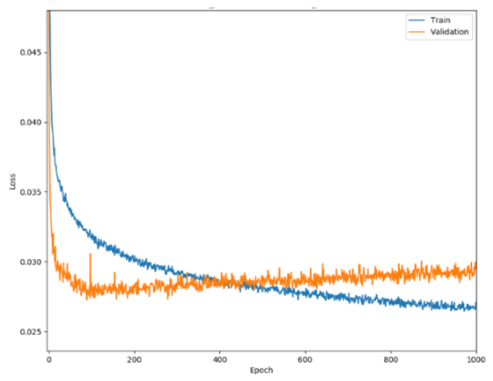
Figure 4.9 DNN_h2 model NSI- loss function and MAE learning curves: epochs =500



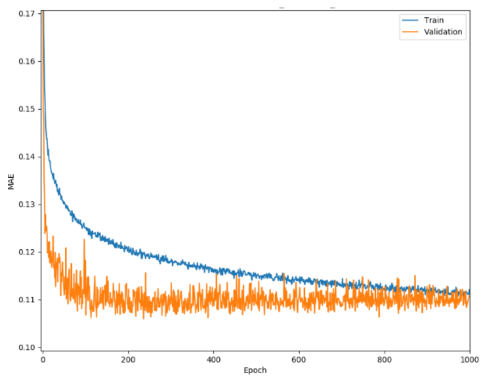
Loss function-FTC



MAE-FTC

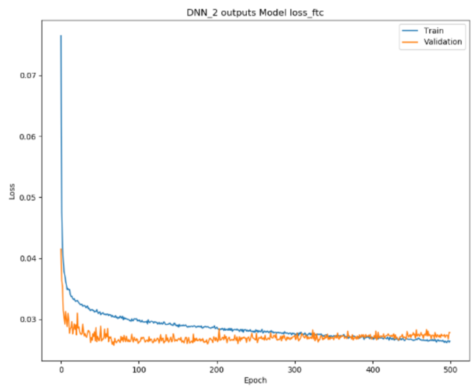


Loss function-1H

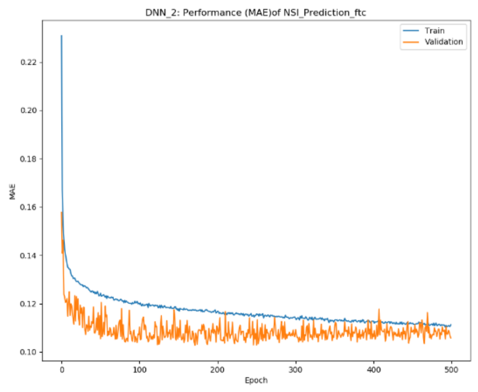


MAE-1H

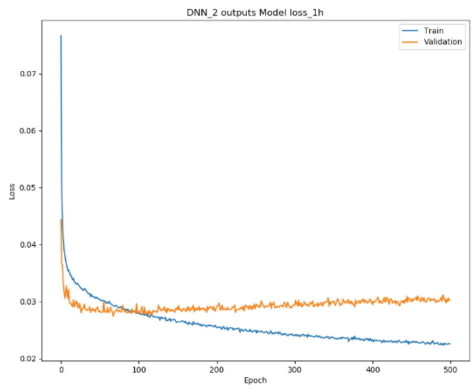
Figure 4.10 DNN_h2 model NSI - loss function and MAE learning curves: epochs =1000



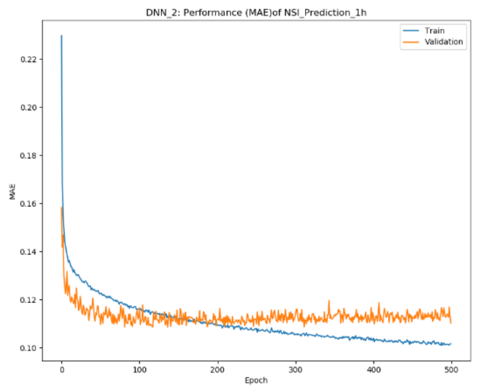
Loss function-FTC



MAE-FTC

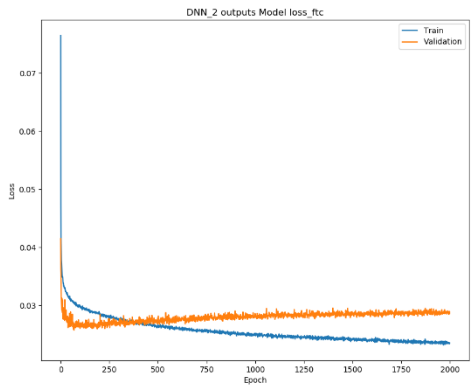


Loss function-1H

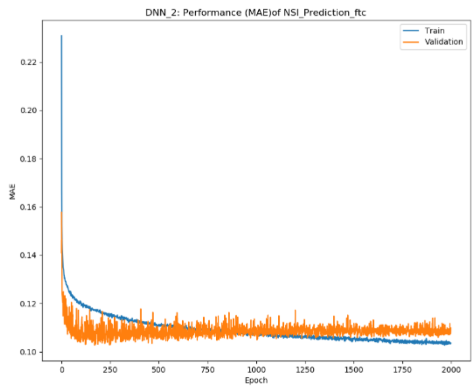


MAE-1H

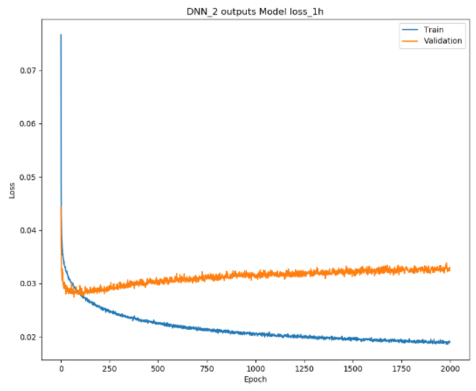
Figure 4.11 DNN_h2T model NSI- loss function and MAE learning curves: epochs =500



Loss function-FTC



MAE-FTC



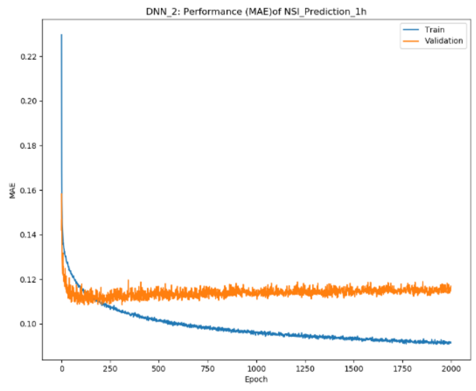
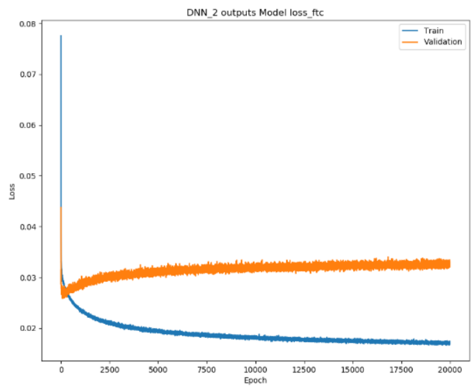
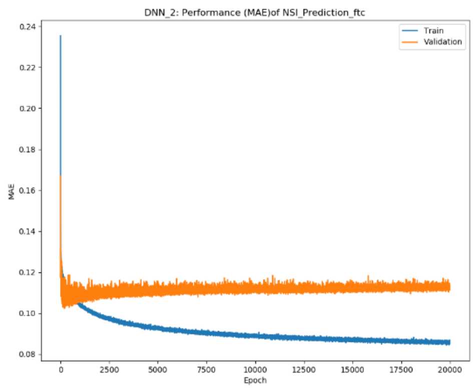


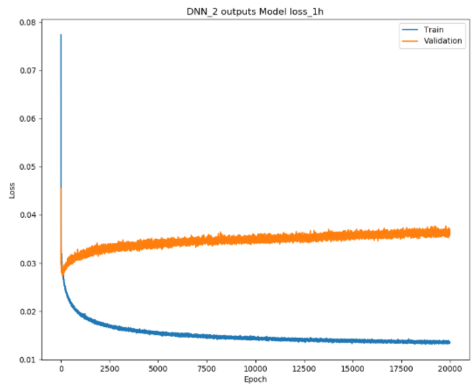
Figure 4.12 DNN_h2T model NSI- loss function and MAE learning curves: epochs =2000



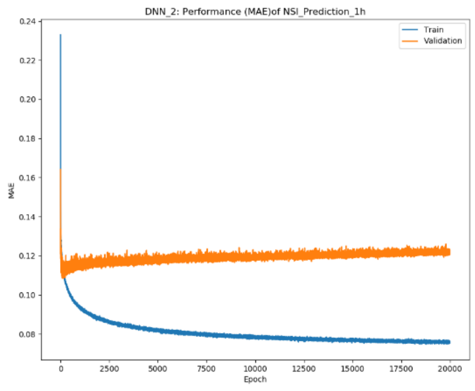
Loss function-FTC



MAE-FTC



Loss function-1H



MAE-1H

Figure 4.13 DNN_h2T model NSI- loss function and MAE learning curves: epochs =20000

Table 4.2 shows the MAE results: the change of epochs for DNN_h2 and DNN_h2T.

Models	**MAE**				
	FTC encoding			1H encoding	
		Train	Val	Train	Val
Single-step prediction: DNN_h2 model					
DNN_h2	**epochs = 500**				
	NSI	0.1209	0.1062	0.1152	0.1098
	epochs = 1000				
	NSI	0.1183	0.1041	0.1121	0.1107
Multi-step prediction: DNN_h2T model epochs = 500					
DNN_h2T	NSI	0.1113	0.1058	0.1017	0.1102
	T	0.0396	0.0162	0.0396	0.0189
Multi-step prediction: DNN_h2T model epochs = 2000					
DNN_h2T	NSI	0.1035	0.1098	0.0914	0.1169
	T	0.0391	0.0174	0.0397	0.0189
Multi-step prediction: DNN_h2T model epochs = 20000					
DNN_h2T	NSI	0.0857	0.1130	0.0754	0.1211
	T	0.0387	0.0176	0.0392	0.0188

As seen previously, DNN_h2T model have decreased to a point of stability where the training and validation curves indicated a behavior of a much better fitted learning curves when the number of epochs increased by increasing the training time that validation loss is improved at about epoch 94, that is, the model would be expected overfitted on epochs 20000 trainings. If testing occurred at the minimum validation loss score, then epochs 500 would be chosen.

Whereas the other learning curves are still having a very narrow gap between the training and validation to indicated that a sign of underfitting may not exist, we would argue that our minimal investigations have attempted to improve and alleviate the underfitting problem. However, the loss functions of our models (DNN_h2 & DNN_h2T) are almost lower on the training set than the validation set which indicates a minimal sign of overfitting in the 1H encoding. We compared the minimum validation errors overall training with the previous initial experiments and investigations to the last minimum validation errors of our DNN_h2T model training on the 500 epochs to find out better results, in which obtained convergent values of the MAE values. However, since the DNN_h2T is the most complex model in our study, we can be argued that the trained models were not suffering or behaving of the underfitting. However, the activation functions are kept as mentioned earlier (subsection 3.6.3) during these investigations.

4.2.2 Change the Number of Nodes

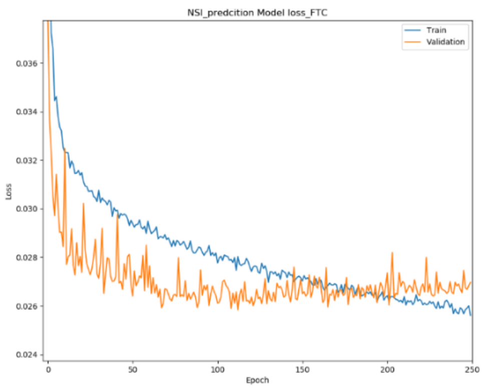
We conducted more investigations to tackle the overfitting/underfitting problems which may reduce them accordingly. That is, changing the number of nodes in our models hidden layers are used as the base experiments end out. We would expect that as the number of nodes is increased, this may increase the capacity of the models and let the models to learn the training set better and vice versa. Likewise, we kept and used the base experiment number of epochs settings on epochs 250, batch size set to 128, and set the dropout layer rate value to (0.3) [22].

The number of nodes are used as follows:

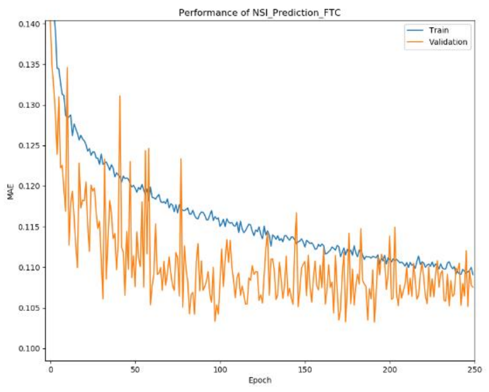
a) Double the Nodes (i.e. hidden layer 1 set to 256, and hidden layer 2 set to 64).

b) Half the Nodes (i.e. hidden layer 1 set to 64, and hidden layer 2 set to 16).

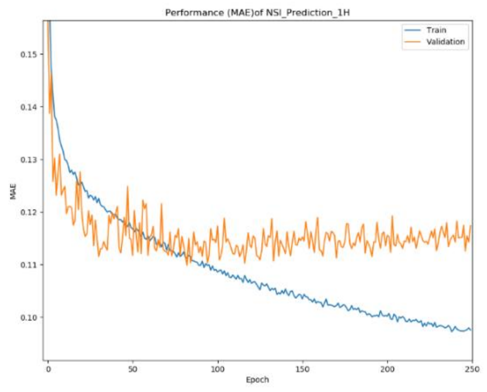
On the following pages, we conducted double nodes experiments on the hidden layers (h1=256 and h2=64) of the DNN_h2 and DNN_h2T models, where figures 4.14, and 4.15 are shown the NSI loss functions and the performance (MAE) NSI values in both encodings FTC and 1H respectively. For the DNN_h2 model, the NSI loss functions and MAE values are depicted that an overfitting occurs around epoch 50 in the 1H encoding, whereas the sign of overfitting takes place after epoch =200 in the loss function in FTC encoding and tends to be slightly fitted for the MAE values within the same FTC encoding where these are a sign of a well performance to the training set. That is, it would be illustrated the learning curves have no implicit overfitting and have no gap between training and validation performance results in the FTC encoding, whereas decreased in generalization errors. Unlike this, there is an overfitting does have the 1H encoding that indicates an overcapacity may be affected the learning rate in an acceptable value if the number of epochs increased and again this is a sign of a well performance to lowering the training errors. However, the most complex model (DNN_h2T model) is shown NSI loss functions that have the same overfitting issue in both encodings, but has a significant sign of a well-fitted trend of the MAE values in the FTC and a sign of overfitting in 1H.



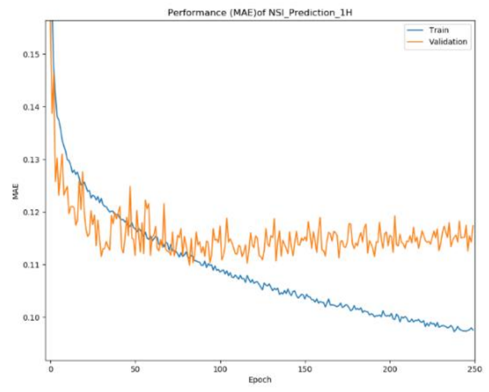
Loss function-FTC



MAE-FTC

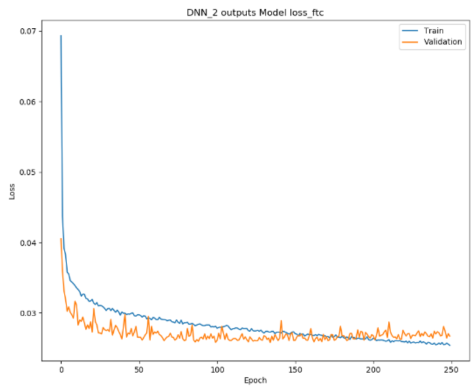


Loss function-1H

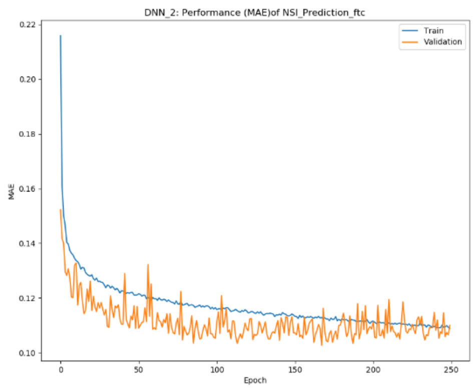


MAE-1H

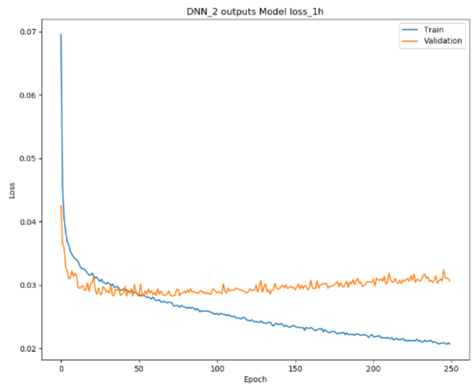
Figure 4.14 DNN_h2 model NSI learning curves: batch=128, double of nodes.



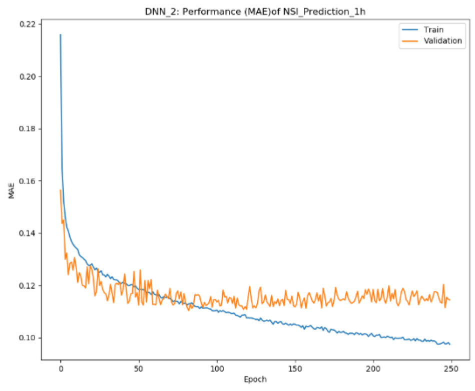
Loss function-FTC



MAE-FTC



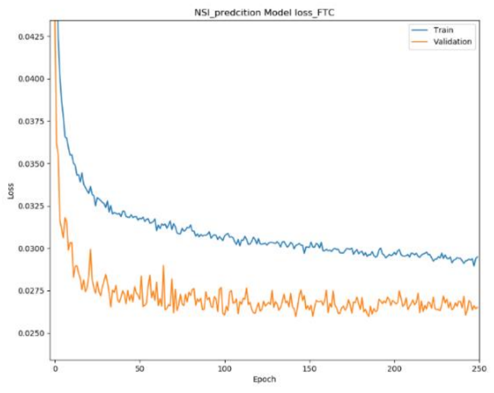
Loss function-1H



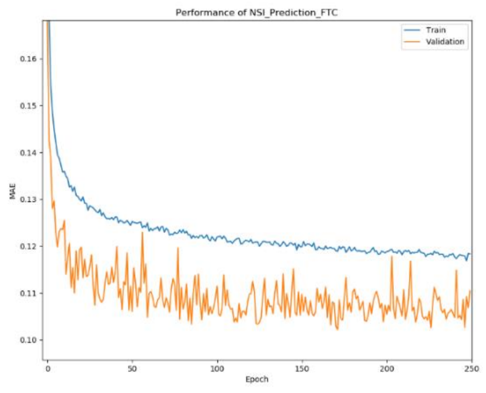
MAE-1H

Figure 4.15 DNN_h2T model NSI learning curves: batch=128, double of nodes.

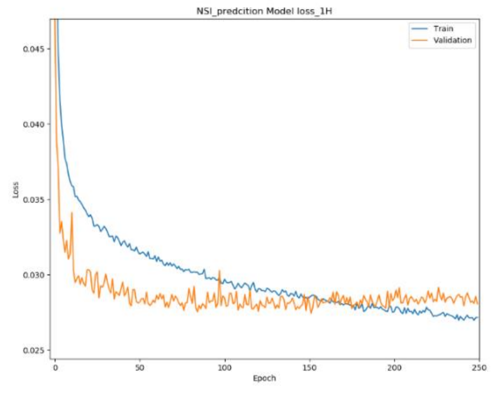
However, we conducted the half nodes experiments for the hidden layers (h1=64 and h2=16) of both DNN_h2 and DNN_h2T models. Figure 4.16 is shown the learning curves of the DNN_h2 model which depicted that MAE values are still suffering from the sign of underfitting behavior in both encodings, whereas the loss function in 1H encoding got an overfit around epoch 180.



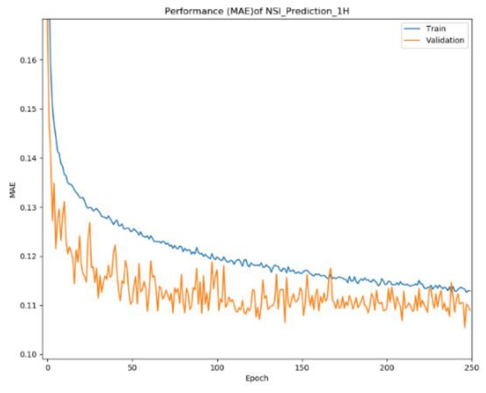
Loss function-FTC



MAE-FTC



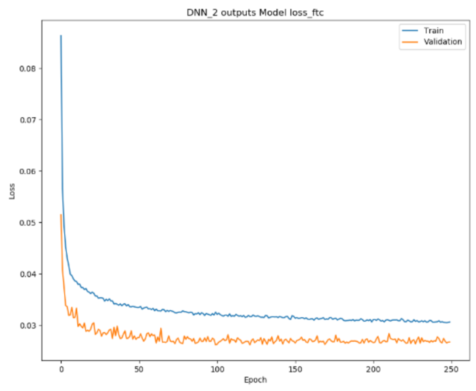
Loss function-1H



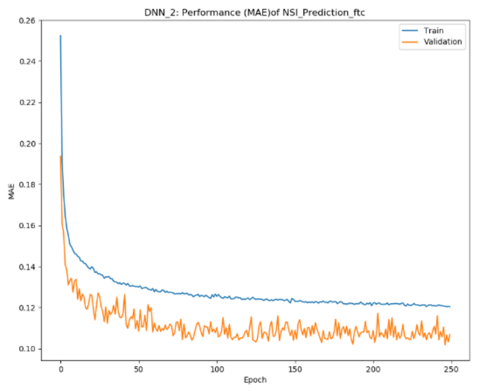
MAE-1H

Figure 4.16 DNN_h2 model NSI loss function and MAE learning curves: batch=128, half of nodes.

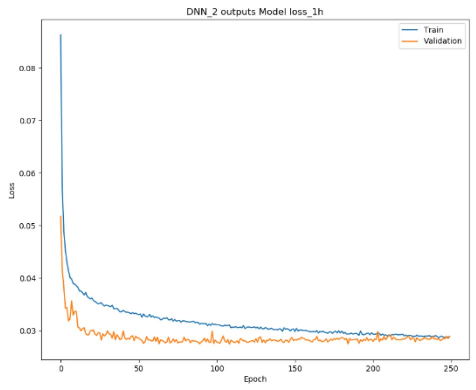
Figure 4.17 is shown the DNN_h2T model results are illustrated approximately the loss function behaves much less values compared to our baseline values and convergent at epoch 240 in 1H encoding. Table (4.3) shows the MAE of the change number of nodes: double and half nodes.



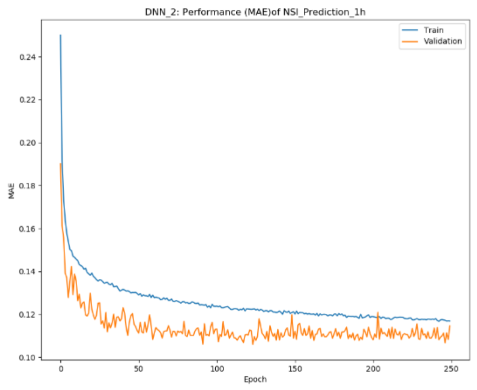
Loss function-FTC



MAE-FTC



Loss function-1H



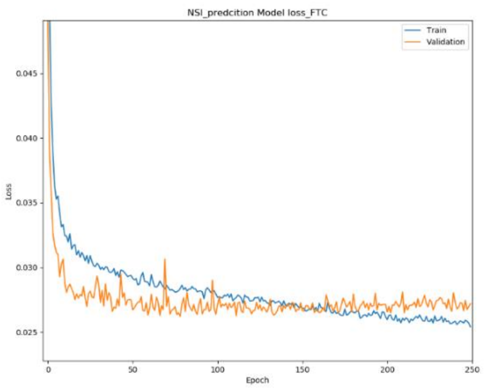
MAE-1H

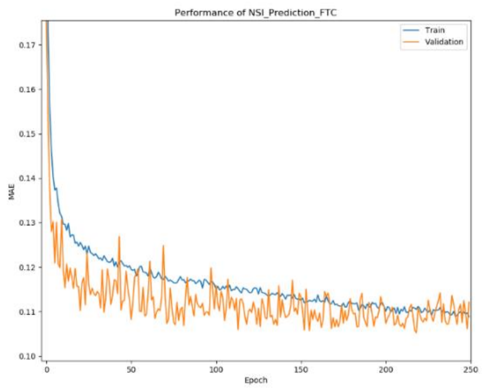
Figure 4.17 DNN_h2T model NSI loss function and MAE learning curves: batch=128 half of nodes.

Table (4.3) shows the MAE results for the change number of nodes: double and half nodes.

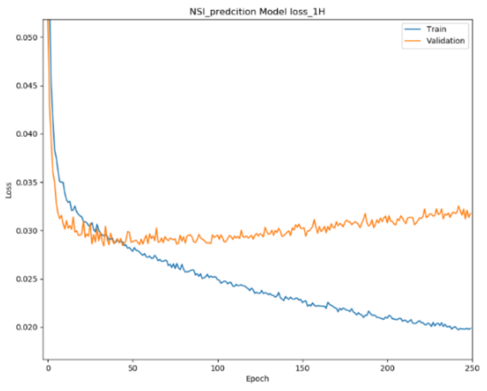
Models		**MAE**							
		epochs = 250 , Dropout(0.3) , batch size= 128, opt=Adam							
		Double nodes: h1 =256 ,h2 =64				**Half nodes: h1 =64, h2 =16**			
		FTC encoding		1H encoding		FTC encoding		1H encoding	
		Train	Val	Train	Val	Train	Val	Train	Val
DNN_h2	NSI	0.1091	0.1075	0.0976	0.1174	0.1183	0.1104	0.1129	0.1089
DNN_h2T	NSI	0.1088	0.1099	0.0973	0.1143	0.1204	0.1068	0.1169	0.1146
	T	0.0311	0.0195	0.0331	0.0179	0.0498	0.0221	0.0511	0.0220
epochs = 250 , Dropout(0.3) , batch size= 512, opt=Adam									
		Double nodes: h1 =256 ,h2 =64				**Half nodes: h1 =64, h2 =16**			
		FTC encoding		1H encoding		FTC encoding		1H encoding	
		Train	Val	Train	Val	Train	Val	Train	Val
DNN_h2	NSI	0.1089	0.1121	0.0947	0.1160	0.1175	0.1079	0.1123	0.1132
DNN_h2T	NSI	0.1101	0.1111	0.0991	0.1195	0.1227	0.1083	0.1184	0.1124
	T	0.0303	0.0166	0.0316	0.0184	0.0502	0.0210	0.0501	0.0236

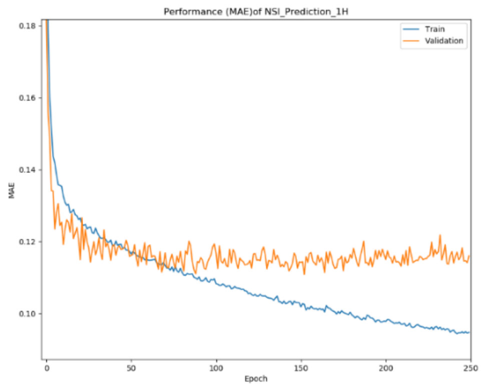
We repeated the previous experiments (subsection 4.2.2) by changing the **batch size to 512**, and kept epochs = 250, and dropout layer rate value (0.3). Thus, on the following pages figures 4.18, and 4.19 are illustrated the DNN_h2 and DNN_h2T models learning curves are taken the double-nodes respectively. Also, figures 4.20, and 4.21 are shown learning curves for the DNN_h2 and DNN_h2T models are taken the half-nodes (h1=64, and h2=16) respectively. The increasing of batch size for given models resulted a comparatively sign of well performance to lowering the training values with some behaviors towards preventing the overfittings in the NSI-loss function and NSI-MAE values, which may be reduced this problematic issue or decided to stop the training.





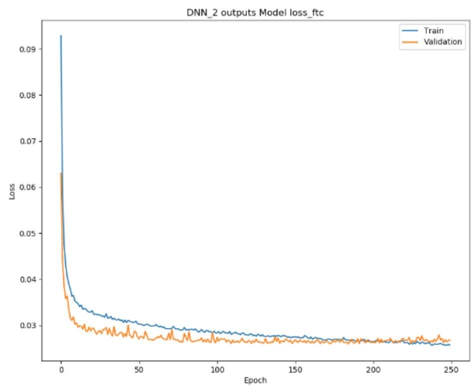
Loss function-FTC MAE-FTC



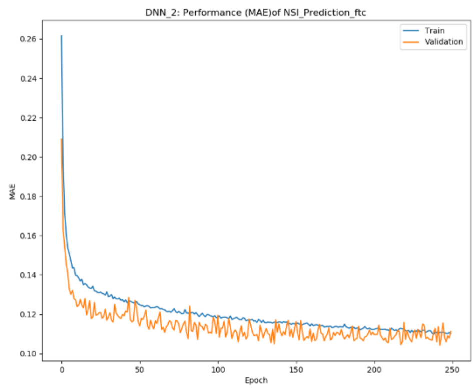


Loss function-1H MAE-1H

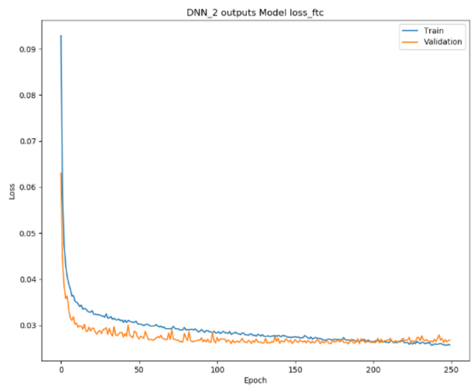
Figure 4.18 DNN_h2 model NSI loss function and MAE learning curves: batch=512, double-nodes



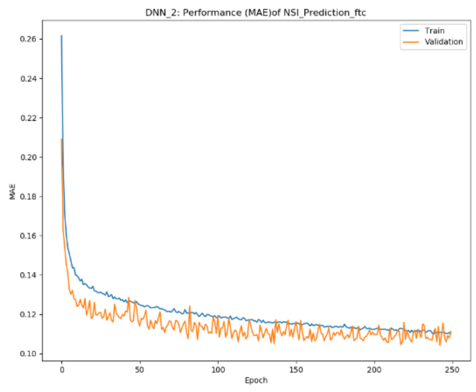
Loss function-FTC



MAE-FTC

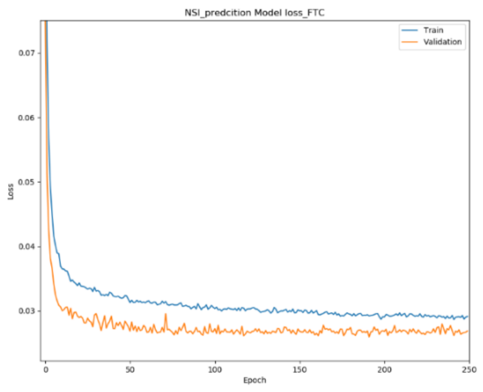


Loss function-1H

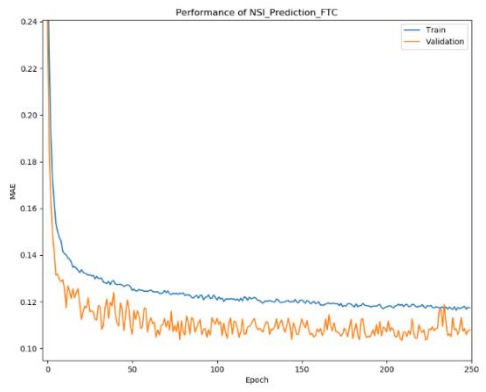


MAE-1H

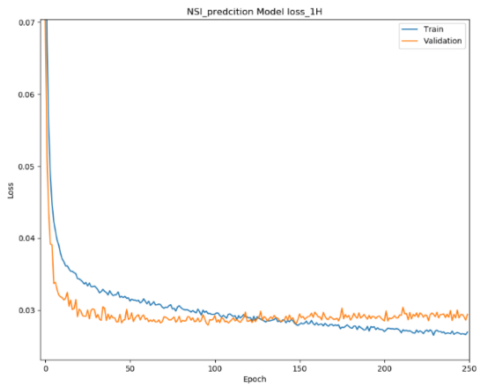
Figure 4.19 DNN_h2T model NSI loss function and MAE learning curves: batch=512, double-nodes.



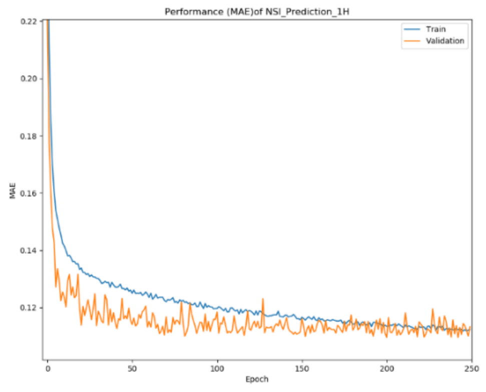
Loss function-FTC



MAE-FTC

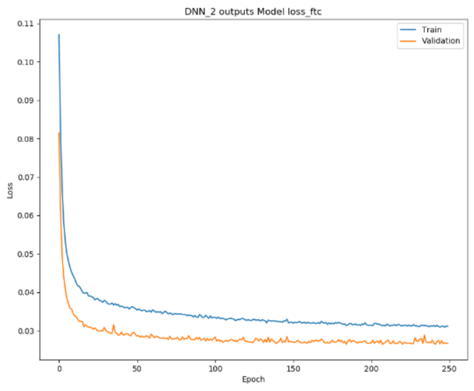


Loss function-1H

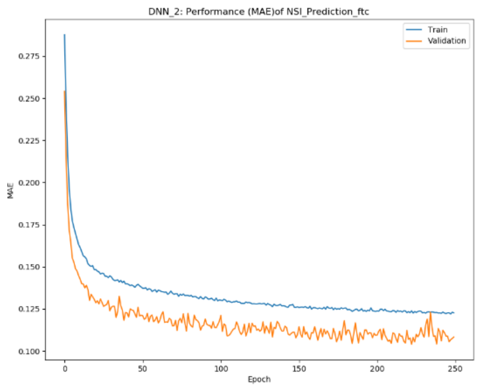


MAE-1H

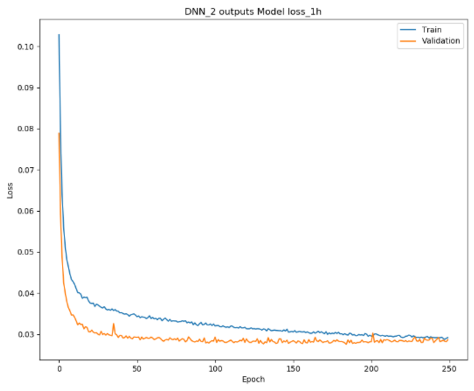
Figure 4.20 DNN_h2 model NSI loss function and MAE learning curves: batch=512, half-nodes.



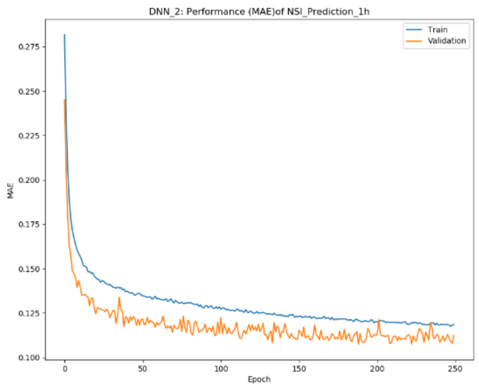
Loss function-FTC



MAE-FTC



Loss function-1H



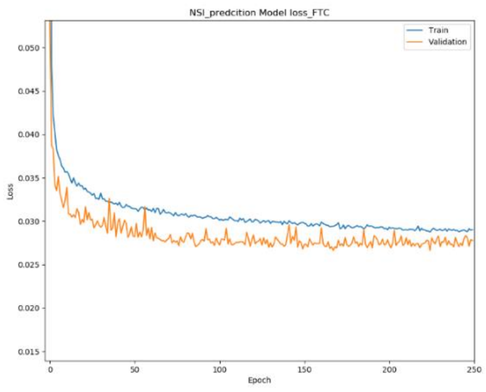
MAE-1H

Figure 4.21 DNN_h2T model NSI loss function and MAE learning curves: batch=512, half-nodes.

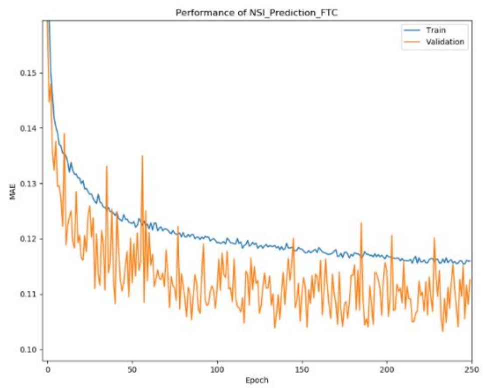
4.2.3 L2 regularization

We conducted more investigation experiments to tackle the overfitting problem by improving the NSI-loss function, which we used L2 regularization for our outputs earlier. That is, added L2 regularizer to the hidden layer 2 (h2) in both the DNN_h2 and DNN_h2T models used a default value (0.01) may behave more preventive against overfitting. We kept the settings of the previous experiment's values: dropout layer rate value (0.3), epochs=250 of both batch sizes 128 and 512. Whereas chose to use the double number of nodes option, which has a sign of highly overfitting values seen previously and had the highest capacity models (i.e. highest number of nodes). However, the results are continued to search for signs of a well-fitted learning curves of performance for predicting the NSI and may reach the benefit of using an intermediate metrological feature T in the DNN_h2T model.

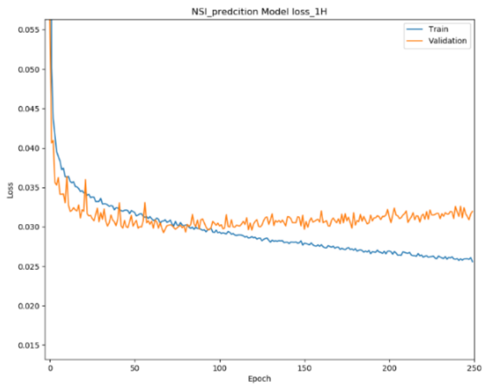
On the next pages, figures 4.22, 4.23, 4.24 and 4.25 are shown the DNN_h2 and DNN_h2T learning curves of both NSI- loss function and performance (MAE) using the double nodes (h1=256, and h2=64) with L2 regularizer of both batch sizes 128 and 512 in both FTC and 1H respectively. Table (4.4) shows the MAE results of the double nodes with L2 regularization.



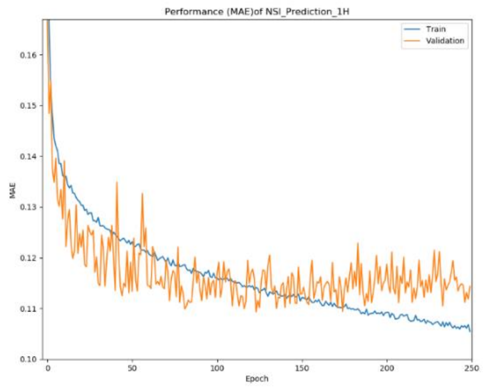
Loss function-FTC



MAE-FTC

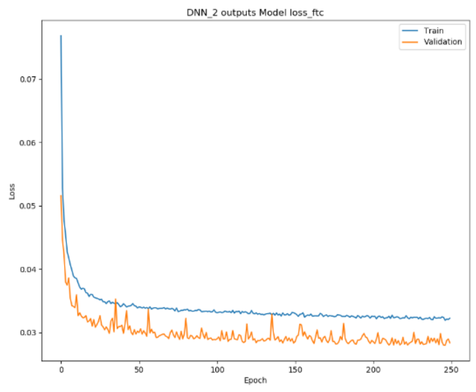


Loss function-1H

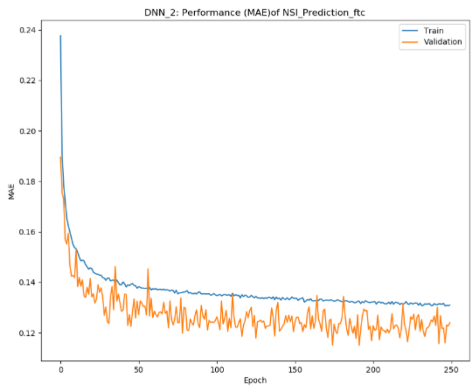


MAE-1H

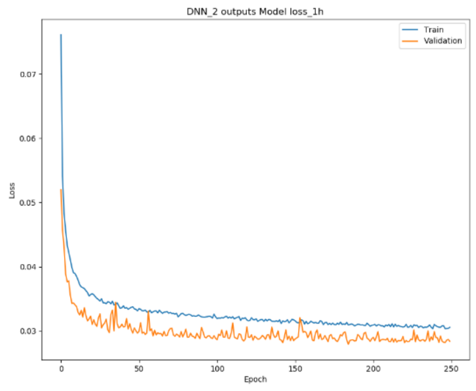
Figure 4.22 DNN_h2 model NSI-loss function and MAE: batch= 128, double-nodes with L2



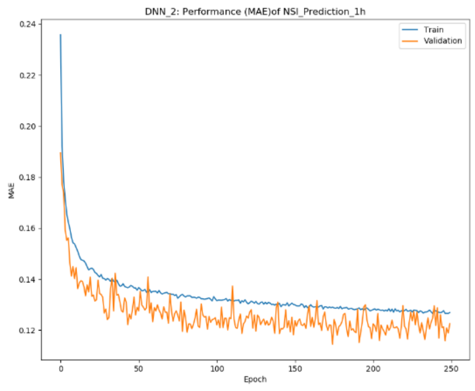
Loss function-FTC



MAE-FTC

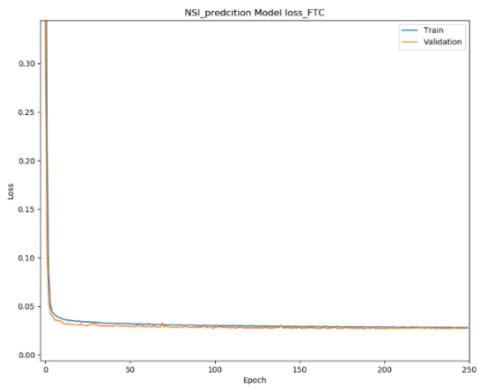


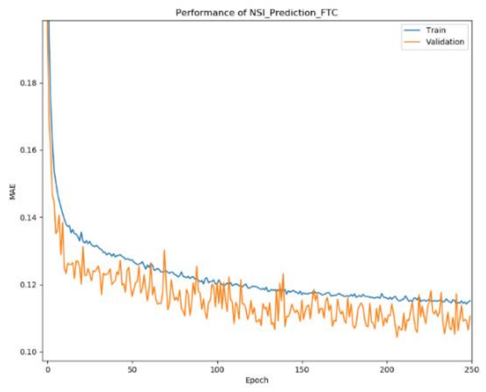
Loss function-1H



MAE-1H

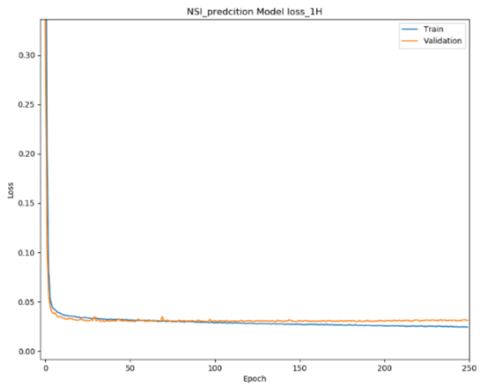
Figure 4.23 DNN_h2T model NSI-loss function and MAE: batch=128, double-nodes with L2

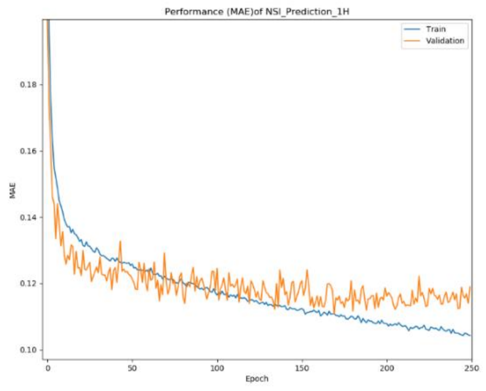




Loss function-FTC

MAE-FTC

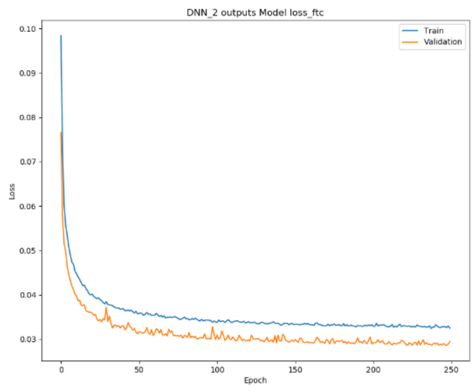




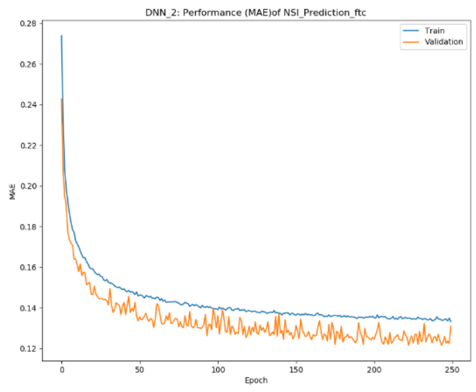
Loss function-1H

MAE-1H

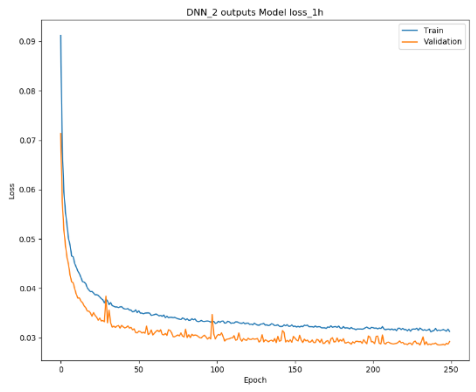
Figure 4.24 DNN_h2 model NSI-loss function and MAE: batch=512, double-nodes with L2



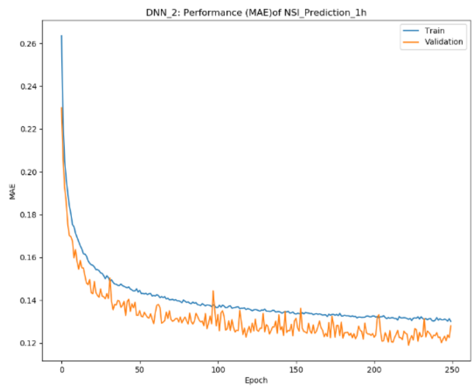
Loss function-FTC



MAE-FTC



Loss function-1H



MAE-1H

Figure 4.25 DNN_h2T model NSI-loss function and MAE: batch=512, double-nodes with L2

Table (4.4) shows the MAE results of the double nodes with L2 regularization.

Models	**MAE**				
	epochs = 250, Dropout (0.3), batch size= 128, opt=Adam Double nodes: h1 =256 ,h2 =64 with L2				
	FTC encoding			1H encoding	
		Train	Val	Train	Val
DNN_h2	NSI	0.1160	0.1126	0.1054	0.1144
DNN_h2T	NSI	0.1310	0.1239	0.1269	0.1226
	T	0.0393	0.0252	0.0387	0.0266
epochs = 250, Dropout (0.3), batch size= 512, opt=Adam Double nodes: h1 =256 ,h2 =64 with L2					
DNN_h2	NSI	0.1151	0.1106	0.1043	0.1189
DNN_h2T	NSI	0.1333	0.1310	0.1301	0.1280
	T	0.0423	0.0295	0.0415	0.0292

To summarize and compare all the MAE results of our experiments throughout training models (i.e. the base experiment MAE among other values), the tables are shown in appendix F, which are illustrated the values of each model and conducted investigation.

4.3 Final Evaluation

We conducted the evaluation experiment from our saved models which have the NSI minimum validation errors in both encodings by which we used the test set (i.e. Testing dataset: one-year (2019), section 3.3) on a given training epoch that corresponding to the minimum scores of the NSI-validation errors per each model of the previous experiments to obtained the generalization errors. That is, a significant NSI lower error values in are showing signs of better-fitted performance to be generalized. However, the models DNN_h2, DNN_h2T and DNN_h2F have obtained comparatively a fitted to predict the (NSI) associated with/ without an intermediate metrological feature (T) or other metrological features. Table (4.5) is illustrated MAE evaluation values of our saved models used the test set on chosen epochs corresponding to the minimum validation scores. There are several ways to improve the optimally tuned this model in the future work. However, all given models have to have convergent MAE values which are indicated that the evaluation may need study more models like the CNN, and RNN for future work. The models are compared together through exploiting both FTC and 1H encodings are depicted a sign of well-fitted performance, where a significant representation to be recognized values within FTC against the 1H encodings.

Table 4.5 shows the MAE evaluation test experiment from the saved models.

Models	**MAE - Evaluation**			**epoch**
	FTC encoding		1H encoding	
DNN_h0	NSI	0.1315	0.1345	250
DNN_h2	NSI	0.1131	0.1197	1000
DNN_h2T	NSI	0.1158	0.1236	500
	T	0.0165	0.0200	
DNN_h2F	NSI	0.1189	0.1204	250
	T	0.0262	0.0295	

CHAPTER V

CONCLUSIONS

The thesis work has contributed two groups of experiments: single-step and multi-step prediction models to predict the normal solar irradiance (NSI) associated with an intermediate meteorological feature such as (T) and may be other features like RH, W, PW, P, PR. This work utilized part of the [22] works, which considered a real-world timeseries dataset (NREL). We proposed two different time/date transformation encodings: FTC (sine-cosine) and 1H (one-hot) representations.

The dense neural network models are included DNN_h0, DNN_h2, DNN_h2T and DNN_h2F are evaluated in terms of mean absolute error (MAE) and proposed to determine the effectiveness of the models generalization performance. As a result, the comparison shows that these models performance are revealed that the (NSI) has an acceptable model's performance in both FTC and 1H, where associated with an intermediate metrological feature such as (T). Whereas the single-step model: DNN_h0 has got slightly acceptance to find well performance to prediction the (NSI). The proposed FTC encoding has reached out a sign of well-fitted and more stabilized expectations to get a sign of better performance.

Further research in the significance of individual neural networks alongside using both encodings FTC and 1H representations might be used for convolutional neural networks (CNN) is necessary to complete the view of this study assumption and recommended for future studies.

REFERENCES

1. Montgomery, D., Jennings, C., & Kulahci, M. (2008). Introduction to time series analysis and forecasting (Wiley series in probability and statistics). Hoboken, N.J.: Wiley-Interscience.

2. Bisgaard, S., & Kulahci, M. (2011). Time series analysis and forecasting by example (Wiley series in probability and statistics). Hoboken, N.J.: John Wiley & Sons.

3. Kauffman, J., Lee, K., & Lee, K. (2013). Handbook of sustainable engineering. Dordrecht ; New York: Springer.

4. Shumway, R., & Stoffer, D. (2017). Time Series Analysis and Its Applications: With R Examples. In Time Series Analysis and Its Applications (4th ed. 2017). Springer International Publishing AG. https://doi.org/10.1007/978-3-319-52452-8

5. Jun Lu, Shaobo Liu, Qing Wu, & Qinru Qiu. (2010). Accurate modeling and prediction of energy availability in energy harvesting real-time embedded systems. 469–476. https://doi.org/10.1109/GREENCOMP.2010.5598280

6. Hejase, H., & Assi, A. (2012). Time-Series Regression Model for Prediction of Mean Daily Global Solar Radiation in Al-Ain, UAE. ISRN Renewable Energy, 2012, 1–11. https://doi.org/10.5402/2012/412471

7. Cammarano, A., Petrioli, C., & Spenza, D. (2012). Pro-Energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks. 75–83. https://doi.org/10.1109/MASS.2012.6502504

8. Christian Renner. (2013). Solar harvest prediction supported by cloud cover forecasts. In Proceedings of the 1st International Workshop on Energy Neutral Sensing Systems (ENSSys '13). Association for Computing Machinery, New York, NY, USA, Article 1, 1–6.

9. Prema, V., & Rao, K. (2015). Development of statistical time series models for solar power prediction. Renewable Energy, 83, 100–109. https://doi.org/10.1016/j.renene.2015.03.038

10. Sharma, A., & Kakkar, A. (2018). Forecasting daily global solar irradiance generation using machine learning. Renewable and Sustainable Energy Reviews, 82(P3), 2254-2269.

11. Gamboa, J. (2017). Deep Learning for Time-Series Analysis.

12. Kilimci, Z., Akyuz, A., Uysal, M., Akyokus, S., Uysal, M., Atak Bulbul, B., & Ekmis, M. (2019). An Improved Demand Forecasting Model Using Deep Learning Approach and Proposed Decision Integration Strategy for Supply Chain. Complexity (New York, N.Y.), 2019, 1–15. https://doi.org/10.1155/2019/9067367

13. Pearson, R., Neuvo, Y., Astola, J., & Gabbouj, M. (2016). Generalized Hampel Filters. EURASIP Journal on Advances in Signal Processing, 2016(1), 1–18. https://doi.org/10.1186/s13634-016-0383-6

14. Raghunathan, V., Kansa,l A., Hsu, J., Friedman, J., and Srivastava, M. (2005) " Design Considerations for Solar Energy Harvesting Wireless Embedded Systems." InProc.ACM/IEEE Intl. Symp. on Information Processing in Sensor Networks, IPSN.

15. Brunelli, D., Moser, C., Thiele, L., and Benini, L. (2009) " Design of a Solar-Harvesting Circuit for Batteryless Embedded Systems." IEEE Transactions on Circuits and Systems I (TCAS-I): Regular Papers, 56(11)

16. Lu, C., Raghunathan, V., & Roy, K. (2011). Efficient Design of Micro-Scale Energy Harvesting Systems. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 1(3), 254–266. https://doi.org/10.1109/jetcas.2011.2162161

17. Kansal, A., Hsu, J., Zahedi, S., & Srivastava, M. (2007). Power management in energy harvesting sensor networks. ACM Transactions on Embedded Computing Systems (TECS), 6(4), 32–es. https://doi.org/10.1145/1274858.1274870

18. Recas Piorno, J., Bergonzini, C., Atienza, D., and SimunicRosing, T. (2009) "Prediction and Management in Energy Harvested Wireless Sensor Nodes" InProc. Intl. Conf. on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronics Systems Technology, VITAE

19. Bergonzini, C., Brunelli, D., & Benini, L. (2010). Comparison of energy intake prediction algorithms for systems powered by photovoltaic harvesters. Microelectronics Journal, 41(11), 766–777. https://doi.org/10.1016/j.mejo.2010.05.003

20. Ali, M., Al-Hashimi, B., Recas Piorno, J., and Atienza, D. (2010) "Evaluation and Design Exploration of Solar Harvested-Energy Prediction Algorithm" InProc. Conf. on Design, Automation and Test in Europe.

21. Cui, S. (2018). Solar energy prediction and task scheduling for wireless sensor nodes based on long short term memory. Journal of Physics: Conference Series, 1074, 012100. doi: 10.1088/1742-6596/1074/1/012100

22. Donghun Lee, & Kwanho Kim. (2019). Recurrent Neural Network-Based Hourly Prediction of Photovoltaic Power Output Using Meteorological Information. Energies (Basel), 12(2), 215–. https://doi.org/10.3390/en12020215

23. Wilson, G. (2016). Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1 [Review of Time Series Analysis: Forecasting and Control, 5th Edition, by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel and Greta M. Ljung, 2015. Published by John Wiley and Sons Inc., Hoboken, New Jersey, pp. 712. ISBN: 978-1-118-67502-1]. 37(5), 709–711. Wiley Subscription Services, Inc. https://doi.org/10.1111/jtsa.12194

24. Kumar, J., Goomer, R., & Singh, A. (2018). Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters. Procedia Computer Science, 125, 676–682. https://doi.org/10.1016/j.procs.2017.12.087

25. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

26. Manaswi, N. (2018). Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition with TensorFlow and Keras. In Deep Learning with Applications Using Python (1st ed.). Apress L. P. https://doi.org/10.1007/978-1-4842-3516-4

27. Pal, A., & Prakash, P. (2017). Practical Time Series Analysis. In Practical Time Series Analysis (1st ed.). Packt Publishing.

28. Lee, C., Lee, J., & Lee, A. (2013). Statistics for Business and Financial Economics (3rd ed. 2013.). Springer New York. https://doi.org/10.1007/978-1-4614-5897-5

29. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.

30. Politis, D. (2015). Model-Free Prediction and Regression: A Transformation-Based Approach to Inference. In Model-Free Prediction and Regression. Springer International Publishing AG.

31. Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. ArXiv, abs/1308.0850.

32. Bendong Zhao;Huanzhang Lu;Shangfeng Chen;Junliang Liu;Dongya Wu. (2017). Convolutional neural networks for time series classification. Journal of Systems Engineering and Electronics, 28(1), 162–169. https://doi.org/10.21629/JSEE.2017.01.18

33. SORKUN, M. C., DURMAZ INCEL, Ö., & PAOLI, C. (2020). Time series forecasting on multivariate solar radiation data using deep learning (LSTM). Turkish Journal of Electrical Engineering & Computer Sciences, 28(1), 211–223. https://doi-org.ezproxy.osu-tulsa.okstate.edu/10.3906/elk-1907-218

34. Nair, V., Chatterjee, M., Tavakoli, N., Namin, A., & Snoeyink, C. (2020). Fast Fourier Transformation for Optimizing Convolutional Neural Networks in Object Recognition.

35. Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., & Ng, R. (2020). Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains.

36. Xu. (2018). Understanding training and generalization in deep learning by Fourier analysis.

37. Chitsaz, K., Hajabdollahi, M., Karimi, N., Samavi, S., & Shirani, S. (2020). Acceleration of Convolutional Neural Network Using FFT-Based Split Convolutions.

38. Cort J. Willmott, & Kenji Matsuura. (2005). Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. Climate Research, 30(1), 79–82. https://doi.org/10.3354/cr030079

39. Miller, W. (2013). Statistics and Measurement Concepts with OpenStat (1st ed. 2013.). Springer New York. https://doi.org/10.1007/978-1-4614-5743-5

40. Barde, M., & Barde, P. (2012). What to use to express the variability of data: Standard deviation or standard error of mean? Perspectives in Clinical Research, 3(3), 113–116. https://doi.org/10.4103/2229-3485.100662

41. Wolter, M., Gall, J., & Yao, A. (2018). Sequence Prediction using Spectral RNNs.

42. Chikodili, N., Abdulmalik, M., Abisoye, O., & Bashir, S. (2021). Outlier Detection in Multivariate Time Series Data Using a Fusion of K-Medoid, Standardized Euclidean Distance and Z-Score. In Information and Communication Technology and Applications (pp. 259–271). Springer International Publishing. https://doi.org/10.1007/978-3-030-69143-1_21

43. Chollet, F. (2017). Deep learning with python. Manning Publications.

44. Sklearn.preprocessing. MinMaxScaler — scikit-learn 0.24.1 documentation. (n.d.). scikit-learn: machine learning in Python — scikit-learn 0.16.1 documentation. Retrieved May 6, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

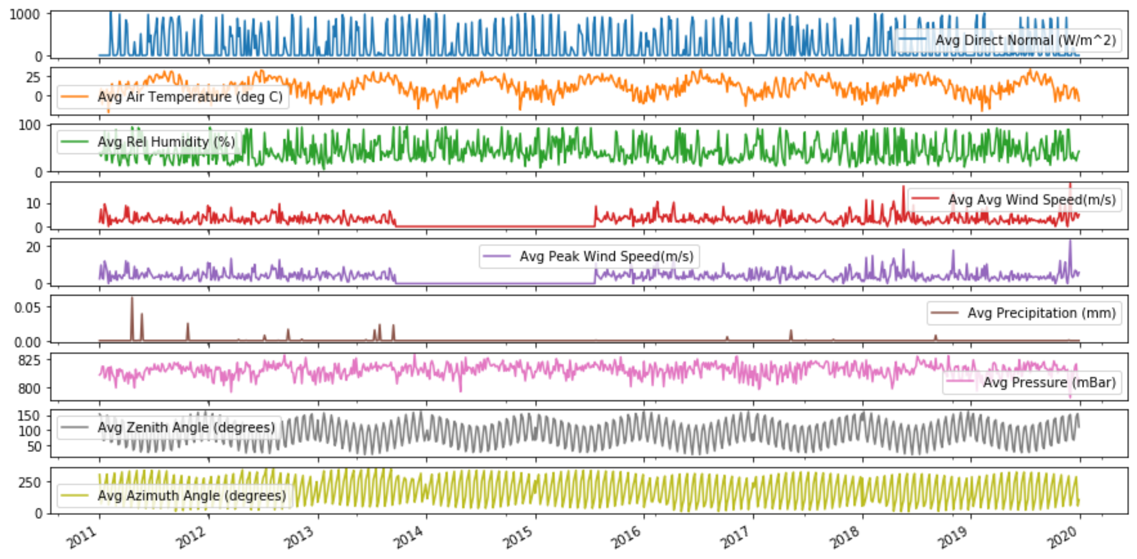
45. Corporate Finance Institute. (2020, May 18). Empirical rule - Overview, formula for standard deviation, uses. https://corporatefinanceinstitute.com/resources/knowledge/other/empirical-rule/

46. Interpolation (scipy.interpolate) — SciPy v1.7.0 manual. *(n.d.). Numpy and Scipy Documentation — Numpy and Scipy documentation.* *https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html*
47. Python numerical methods. *(n.d.).* *https://pythonnumericalmethods.berkeley.edu/notebooks/chapter17.03-Cubic-Spline-Interpolation.html*
48. Dropout: A simple way to prevent neural networks from Overfitting. (n.d.). Journal of Machine Learning Research. https://jmlr.org/papers/v15/srivastava14a.html

APPENDICES

APPENDIX A

Original Datasets



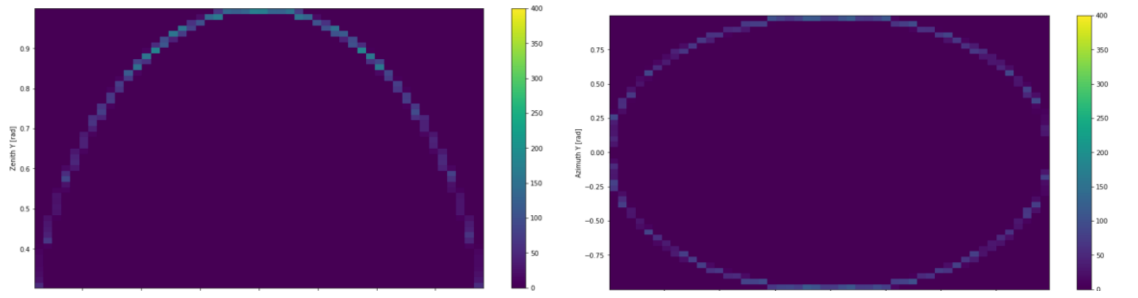
An evolution of the meteorological features recorded overtime of the NREL dataset.

APPENDIX B

Data Configuration

We used the NumPy package as np (version 1.17.4) to calculate the 24 hours of the day per year by using the sine and cosine functions and one-hot encode. Likewise, the angles are converted by using sine and cosine functions. The code below describes the conversion used in the thesis work.

```
def degreeToCosSin(d):
    r = (d/180.0)*np.pi
    return np.sin(r), np.cos(r)
```



**** the distribution of Avg Zenith Angle (degrees) and Avg Azimuth Angle (degrees):Convert to radians ****

```
def dateToDayOfYear(strDate):
    format='%m/%d/%Y'
    date = datetime.datetime.strptime(strDate, format)
    yearStart = date.replace(day=1,month=1)
    dayOfYear = date.toordinal() - yearStart.toordinal()
    return dayOfYear

def dateToCosSin(strDate):
    dayOfYear = dateToDayOfYear(strDate)
    r = dayOfYear*2.0*np.pi /365.0
    return np.sin(r),np.cos(r)

def hourToCosSin(h):
    r = (h/24.0) *2 * np.pi   # 0 to 1 range
    hs = np.sin(r)
```

```
   hc = np.cos(r)
   return hs,hc

def sincostoHour(hs,hc):
   if hs >= 0:
      h = (12.0)* np.arctan2(hs,hc)/np.pi
   else:
       h = ((12.0)* np.arctan2(hs,hc)/np.pi) + 24
   #return round(h)
   return h

def hourToOneHot(h):
   # with h from 0 to 23, inclusuve
   h = int(h)
   oneHotHour = np.zeros((24),dtype=float)
   oneHotHour[h] = 1.0
   return oneHotHour

def onehotToHour(h_oneHot):
   for h in range(len(h_oneHot)):
       if h_oneHot[h]>0:
          return h
   print("invalide One-Hot encoding")

def dateToOneHot(strDate):
   dayOfYear = dateToDayOfYear(strDate)
   oneHotDate = np.zeros((26),dtype=float)
   binSize = 365//26 # want integer division
   binLoc = (dayOfYear // binSize)%26  # want integer division
   oneHotDate[binLoc] = 1.0
   return oneHotDate
```

APPENDIX C

Data Cleansing and Normalization

```
def modOutlier(data, stdScaler = None):
    # modified to change in place
    if stdScaler == None :
        # create transform
        s = StandardScaler(copy=True,with_mean=True,with_std=True)
        z = s.fit_transform(data)
    else:
        # using existing transform
        s = stdScaler
        z = s.transform(data)
    numRows = data.shape[0]
    numCols = data.shape[1]
    for i in range(numRows) :
        for j in range (numCols) :
            if abs(z[i,j]) > 3 :
                if (i == 0 ) :
                    if abs(z[i+1,j]) > 3 :
                       print("Error A: adjacent outliers.  Can not linear interpolate")
                       sys.exit(-1)
                    data[i,j] = data[i+1,j]
                elif i == numRows -1 :
                    if abs(z[i-1,j]) > 3 :
                       print("Error B: adjacent outliers.  Can not linear interpolate")
                       sys.exit(-1)
                    data[i,j] =data[i-1,j]
                else:
                    if  abs(z[i+1,j]) > 3  or abs(z[i-1,j]) > 3 :
                       print("{}, {} : Error C: adjacent outliers.  Can not linear interpolate.
Replacing with z=3 value".format(i,j))
                       #sys.exit(-1)
                       #print("  before : data = ",data[i,j])
                       data[i,j] = s.mean_[j] + math.copysign(3,z[i,j]) * s.scale_[j]
                       #print("   after : data = ",data[i,j])
                    else:
                        data[i,j] = (data[i-1,j]+data[i+1,j])/2
    return s
```

APPENDIX D

Sliding window

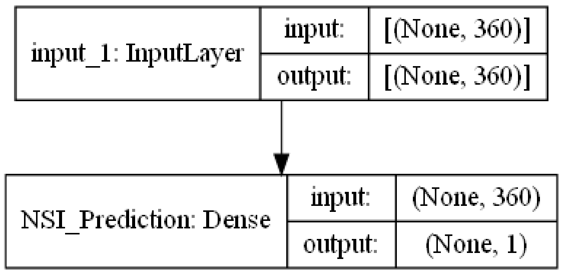
```
import datacfg
odc = datacfg.NRELOriginalDataConfig
ftcdc = datacfg.NREL_FV_FTC_Config
oneHdc = datacfg.NREL_FV_1Hot_Config

import ang

def dayTime(fv, useFTC=True, epsilon=0.1):
  if useFTC:
    h = ang.sincostoHour(fv[ftcdc.HS_LOC],fv[ftcdc.HC_LOC])
  else:
    h = 
ang.onehotToHour(fv[oneHdc.HOUR_START:oneHdc.HOUR_START+oneHdc.HOUR_SIZE])
  if h + epsilon >= 8 and h -epsilon  <= 18 :
    flag  = True
  else:
    flag = False
  return flag

#def sliding_window(arr, windowsize,predLoc=0,useFTC=True,addTemp=False):
# change to place all numeric as y, using all features except the hour and date
def sliding_window(arr, windowsize,predLoc=0,useFTC=True):
  if useFTC :
    dc = ftcdc
  else:
    dc = oneHdc
  ytags  = np.zeros((arr.shape[0],dc.MEASURE_SIZE))  # Temp, NSI prediction
  xdata = np.zeros( (arr.shape[0], windowsize*arr.shape[1]) ) # input windows
  xloc = 0
  for row in range(windowsize, arr.shape[0]-predLoc) :
    if dayTime(arr[row+predLoc],useFTC) : # predicted value is in our day time frame
      xdata[xloc,:] = arr[row-windowsize:row].flatten()
      ytags[xloc,:] = arr[row + 
predLoc,dc.MEASURE_START:dc.MEASURE_START+dc.MEASURE_SIZE]
      xloc += 1
  return xdata[:xloc,:],ytags[:xloc]
```

APPENDIX E

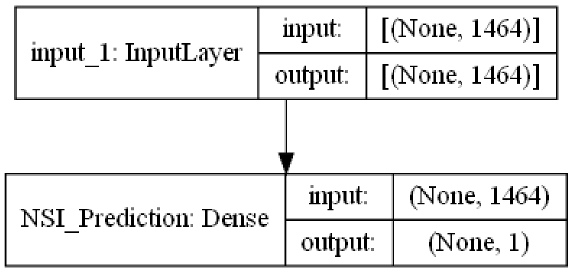
```
def DNN_h0(numInput, numOutput):
    input_layer = tf.keras.layers.Input(shape=(numInput,))
    output_layer = tf.keras.layers.Dense(1, activation='relu', name='NSI_Prediction')(input_layer)
    model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
    return model
def DNN_h2(numInput, numOutput):
    input_layer = tf.keras.layers.Input(shape=(numInput,))
    h1 = tf.keras.layers.Dense(128, activation='relu')(input_layer)
    dropout1 = tf.keras.layers.Dropout(0.3)(h1)
    h2 = tf.keras.layers.Dense(32, activation='relu')(dropout1)
    dropout2 = tf.keras.layers.Dropout(0.3)(h2)
    output_layer = tf.keras.layers.Dense(1, activation='sigmoid',
name='NSI_Prediction')(dropout2)
    model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
    return model
def DNN_h2T(numInput, numOutput):
    input_layer = tf.keras.layers.Input(shape=(numInput,))
    h1 = tf.keras.layers.Dense(128, activation='relu')(input_layer)
    dropout1 = tf.keras.layers.Dropout(0.3)(h1)
    h2 = tf.keras.layers.Dense(32, activation='relu')(dropout1)
    dropout2 = tf.keras.layers.Dropout(0.3)(h2)
    Tout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='Tout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
    a = tf.keras.layers.Concatenate()([Tout, dropout2])
    NSIout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='NSIout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(a)
    model = tf.keras.Model(inputs=input_layer, outputs=[Tout, NSIout])
    return model
def DNN_h2F(numInput, numOutput):
    input_layer = tf.keras.layers.Input(shape=(numInput,))
    h1 = tf.keras.layers.Dense(128, activation='relu')(input_layer)
    dropout1 = tf.keras.layers.Dropout(0.3)(h1)
    h2 = tf.keras.layers.Dense(32, activation='relu')(dropout1)
    dropout2 = tf.keras.layers.Dropout(0.3)(h2)
    RHout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='RHout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
    Wout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='Wout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
    PWout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='PWout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
```

```
    Pout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='Pout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
    PRout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='PRout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
    Tout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='Tout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(dropout2)
    a = tf.keras.layers.Concatenate()([RHout, Wout, PWout, Pout, PRout, Tout, dropout2])
    NSIout = tf.keras.layers.Dense(1, use_bias=True, activation='sigmoid', name='NSIout',
kernel_regularizer=tf.keras.regularizers.l2(0.01))(a)
    model = tf.keras.Model(inputs=input_layer, outputs=[RHout, Wout, PWout, Pout, PRout,
Tout, NSIout])
    return model
```

Models Structures

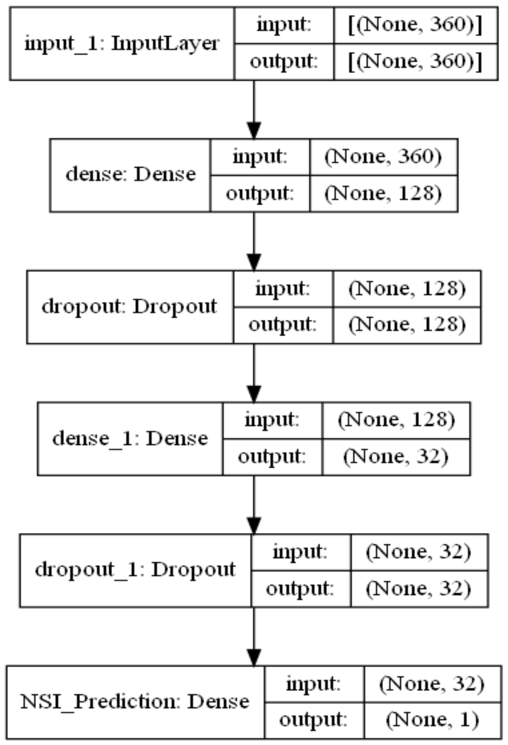


FTC encoding

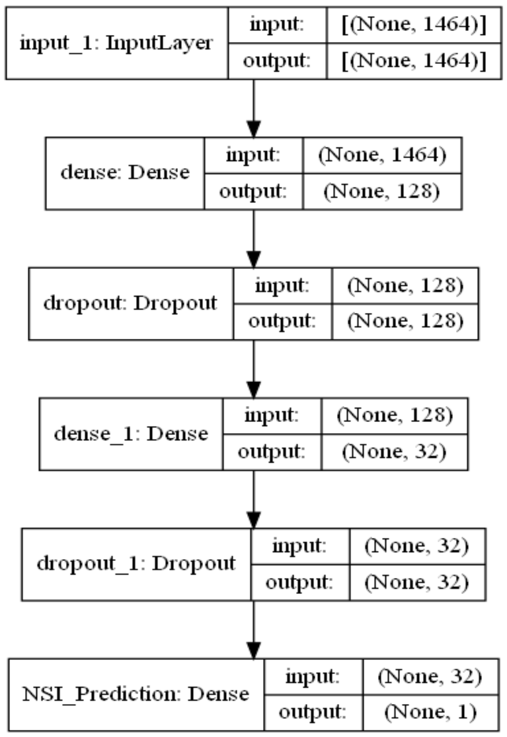


1H encoding

DNN_h0 model structure

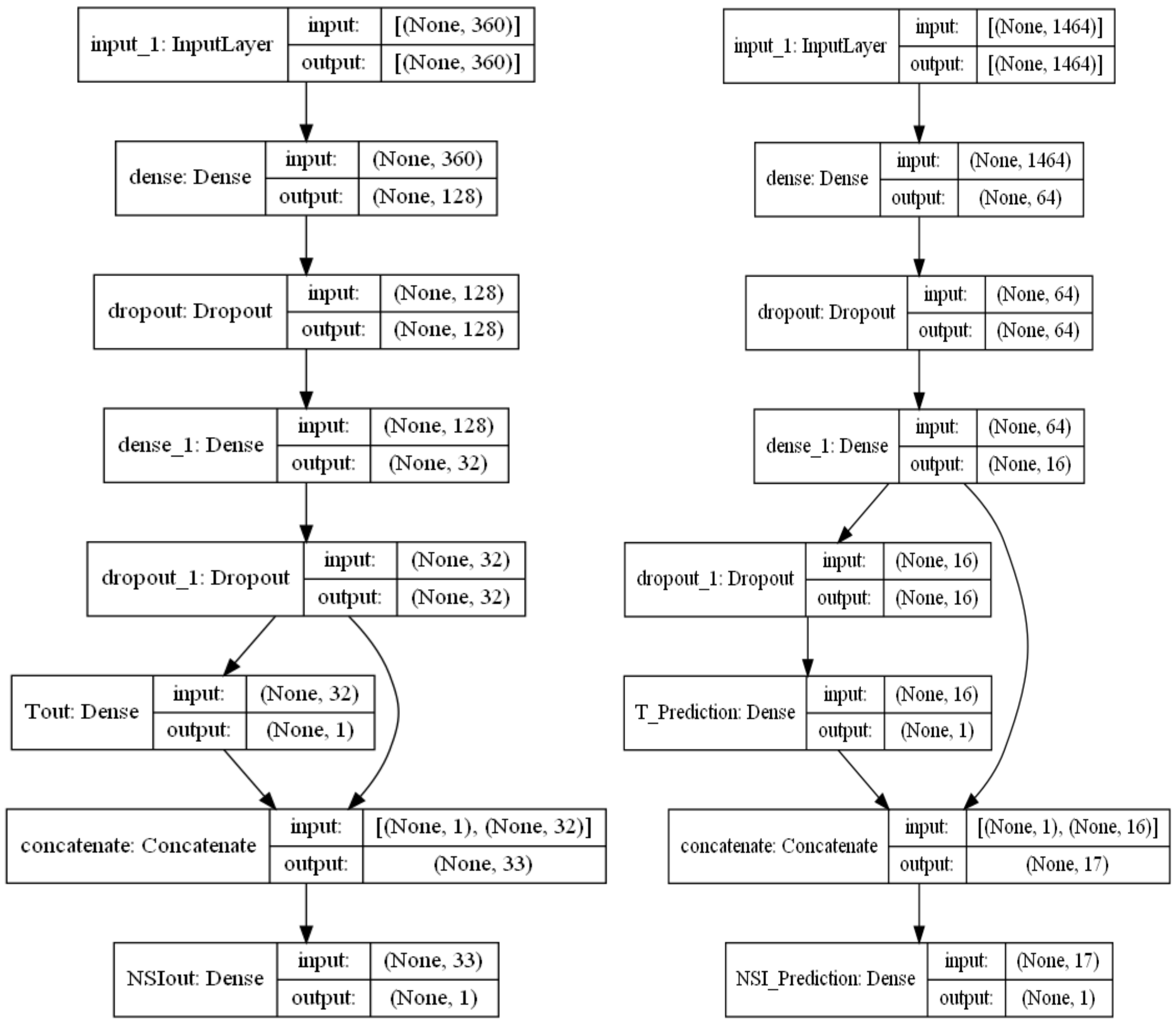


FTC encoding



1H encoding

DNN_h2 model structure



FTC and 1H encodings

DNN_h2T model structure

APPENDIX F

Table (1) shows the MAE results: base experiment and the change of epochs number.

Models	**MAE**				
	base experiment				
	FTC encoding			1H encoding	
		Train	Val	Train	Val
Baseline	NSI	0.1419	0.1365	0.1419	0.1365
Single-step prediction: DNN_h0 and DNN_h2 models epochs = 250 , batch size = 128 , opt=Adam					
DNN_h0	NSI	0.1233	0.1240	0.1244	0.1286
DNN_h2	**h1 =128 ,h2 =32, Dropout (0.3)**				
	NSI	0.1230	0.1076	0.1186	0.1107
	epochs = 500				
	NSI	0.1209	0.1062	0.1152	0.1098
	epochs = 1000				
	NSI	0.1183	0.1041	0.1121	0.1107
Multi-step prediction: DNN_h2T and DNN_h2F models epochs = 250 , batch size = 128 , opt=Adam					
DNN_h2T	NSI	0.1136	0.1074	0.1072	0.1123
	T	0.0373	0.0208	0.0389	0.0237
DNN_h2F	NSI	0.1251	0.1108	0.1216	0.1119
Multi-step prediction: DNN_h2T model epochs = 500 , batch size = 128 , opt=Adam					
DNN_h2T	NSI	0.1113	0.1058	0.1017	0.1102
	T	0.0396	0.0162	0.0396	0.0189
Multi-step prediction: DNN_h2T model epochs = 2000 , batch size = 128 , opt=Adam					
DNN_h2T	NSI	0.1035	0.1098	0.0914	0.1169
	T	0.0391	0.0174	0.0397	0.0189
Multi-step prediction: DNN_h2T model epochs = 20000 , batch size = 128 , opt=Adam					
DNN_h2T	NSI	0.0857	0.1130	0.0754	0.1211
	T	0.0387	0.0176	0.0392	0.0188

Table (2) shows the MAE results for the change number of nodes: double and half nodes.

Models	**MAE**								
	epochs = 250 , Dropout(0.3) , batch size= 128, opt=Adam								
	Double nodes: h1 =256 ,h2 =64					**Half nodes: h1 =64, h2 =16**			
	FTC encoding			1H encoding		FTC encoding		1H encoding	
		Train	Val	Train	Val	Train	Val	Train	Val
DNN_h2	NSI	0.1091	0.1075	0.0976	0.1174	0.1183	0.1104	0.1129	0.1089
DNN_h2T	NSI	0.1088	0.1099	0.0973	0.1143	0.1204	0.1068	0.1169	0.1146
	T	0.0311	0.0195	0.0331	0.0179	0.0498	0.0221	0.0511	0.0220
epochs = 250 , Dropout(0.3) , batch size= 512, opt=Adam									
	Double nodes: h1 =256 ,h2 =64					**Half nodes: h1 =64, h2 =16**			
	FTC encoding			1H encoding		FTC encoding		1H encoding	
		Train	Val	Train	Val	Train	Val	Train	Val
DNN_h2	NSI	0.1089	0.1121	0.0947	0.1160	0.1175	0.1079	0.1123	0.1132
DNN_h2T	NSI	0.1101	0.1111	0.0991	0.1195	0.1227	0.1083	0.1184	0.1124
	T	0.0303	0.0166	0.0316	0.0184	0.0502	0.0210	0.0501	0.0236

Table (3) shows the MAE results of the double nodes with L2 regularization.

Models	**MAE**				
	epochs = 250, Dropout (0.3), batch size= 128, opt=Adam Double nodes: h1 =256 ,h2 =64 with L2				
	FTC encoding			1H encoding	
		Train	Val	Train	Val
DNN_h2	NSI	0.1160	0.1126	0.1054	0.1144
DNN_h2 T	NSI	0.1310	0.1239	0.1269	0.1226
	T	0.0393	0.0252	0.0387	0.0266
epochs = 250, Dropout (0.3), batch size= 512, opt=Adam Double nodes: h1 =256 ,h2 =64 with L2					
DNN_h2	NSI	0.1151	0.1106	0.1043	0.1189
DNN_h2 T	NSI	0.1333	0.1310	0.1301	0.1280
	T	0.0423	0.0295	0.0415	0.0292

VITA

Fadhil A. Ali Alsahlanee

Candidate for the Degree of

Master of Science

Thesis: ANALYSIS OF TIME SERIES FORECASTING IN APPLICATION TO SOLAR ENERGY HARVEST

Major Field: Computer Science

Biographical:

Education:
Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in May, 2022.

Completed the requirements for the Bachelor of Science major in Physics and a minor in computer science at Oklahoma State University, Stillwater, Oklahoma in July, 2017.