Multivariate Time-Series Weather Forecasting

Tejas Dinesh Patil, Yeswanth Reddy Velapalem
School of Information Studies, Syracuse University

Dr. Chilukuri K. Mohan
Syracuse University

Abstract

Forecasting is one of the most important applications in deep learning and machine learning. Various techniques such as AR, MA, ARIMA etc. are used in the forecasting of time series. Our goal here is to use the Artificial Neural Networks for temperature forecasting. We have a total of 14 input parameters related to weather. Thus, along with univariate predictions, we can also make multivariate predictions using historical data from multiple variables. The purpose of this project is to incorporate and compare different forms of neural networks based on the results achieved. These architectures will be compared on the basis of the measured loss (mean squared error) and the computational effort needed to train neural network models.

1. Introduction

The dataset that we are working on for our project is the weather time series data obtained by a research institute called 'Max Planck Institute of Biogeochemistry' based in Jena, Germany. This institute studies human interaction with natural environments and how they influence and react to global warming and environmental change. We intend to build on their research by integrating the predictive powers of artificial neural networks and predicting the temperature values for that region. The dataset provided by the institute is a multivariate time series dataset. A 'csv' file formatted data can be downloaded using the following link:
Dataset Link

The data set consists of 14 variables and almost 420K observations. Each record in the data set is the weather data reported after 10 minutes. Weather attributes like temperature, pressure etc. do not change much in the space of 10 minutes. It was then determined to sample hourly values from the data collection. Taking only the hourly values from the data, the size of the dataset reduced from 420K observations to 70K observations without much loss of information.
A detailed analysis of the statistical definition of the data was conducted to explain the data. Adding to that some visual representations have been plotted to explain shifting trends and data correlation. It has been found that the variables in the data have different units and function on a different scale relative to each other. For example, the temperature (Kelvin) values are between 250.85 K and 311.21 K. While the 'Pressure' values are between 913.6 mbar and 1015.29 mbar. The data was tested for missing values and no missing values were found in the data.
The plot diagram for each of the numerical variables in the data was plotted. Boxplot helps to detect the presence of outliers in the data and to classify the distribution of

variables. Outliers have been found in almost all variables. In particular, 'WindSpeed' and 'MaxWindSpeed' had significant outliers with negative values. The distribution of 'RelativeHumidity' and 'VapPressDef' was distorted, whereas the distribution of other variables was roughly normal.

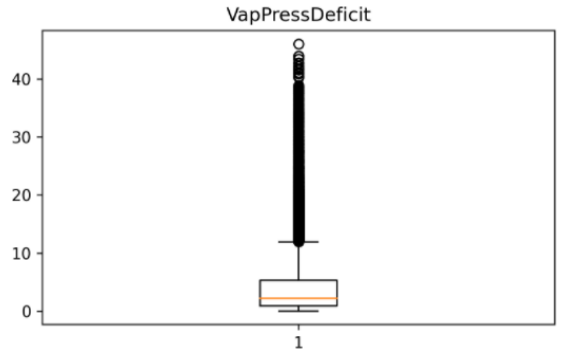

Fig 1. Box plot for 'VapPressDeficit'

Since these are weather-related data and the values were recorded by a device, there is a possibility that outliers may be true values. We chose to use the IQR (Interquartile Range) approach to handle only extreme outliers. All values greater than 'UC' have been reduced to 'UC' and all values smaller than 'LC' have been raised to 'LC' where,

UC = (Quartile: 3) + (3 * IQR);
LC = (Quartile: 1) - (3 * IQR) and
IQR = (Quartile: 3) - (Quartile: 1)

Time-trendline analysis was conducted using numerical variables to explain their behavior over time. Since this is weather-related data, all variables tend to follow the seasonal trend except for the 'WindDirection' and the wind speed variables. In the case of wind speed variables, the speed variance is dominated by the extreme negative values present in the data. WindDirection has constant values over time, and there is no pattern or seasonality.

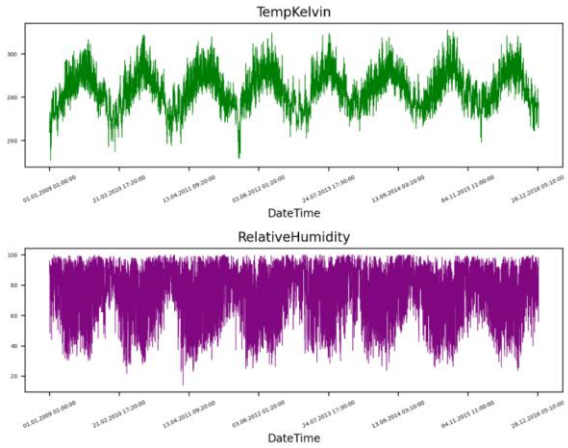

Fig 2. Time-trendline analysis of 'TempKelvin' and 'RelativeHumidity' (Top to Bottom)

A matrix of correlation was plotted to find the correlation of each variable with each other. First, we observed the relationship between the target variable and all other variables. 'Pressure,' 'WindSpeed,' 'MaxWindSpeed' and 'WindDirDegrees' have very poor correlations with 'TempKelvin.' These variables can be either eliminated or modified. Multicollinearity tests were also carried out where it was found that 'TempCelsius' and 'TempKelvin' had a correlation of 1.
Also, 'SpecificHumidity,' 'WaterVapConc' and 'VaporPressure' had a complete correlation between them. Therefore, 'TempCelsius,' 'WaterVapConc' and 'VaporPressure' have been deleted to eliminate data redundancy.
WindSpeed and MaxWindSpeed had negative values of 9999.0. Since speed cannot have negative values, these negative values had been imputed with 0. After taking 'WindSpeed' as the magnitude and 'WindDirDegree' as the direction (radian), wind vectors were created.
X component = WindSpeed * Cos(direction)
Y component = WindSpeed * Sin(direction)

The same steps were followed for 'MaxWindSpeed'. Finally, the 'DateTime' variable was set as the index of the data frame.

2. Generating Sequences from data

Three types of forecasting problems were addressed in this project:

1] Univariate single value forecasting
2] Multivariate single value forecasting
3] Multivariate sequence to sequence forecasting

These three types of processes require different data preprocessing for sequence generation.

2.1 Univariate single value forecasting:

This is where we only use the 'TempKelvin' variable as input and output data. This variable is loaded into a separate data frame and then 80 percent-20 percent of the train-test split is carried out. 'MinMaxScaler' from the sklearn preprocessing library is used to scale the data. The scaling object was built over the train data and used to transform both the train and the test data. Test and train data were re-combined into a data frame.
Thus, a function was written to convert a data frame to a 'numpy' array and generate sequences. Then the size of the train data set is specified which needs to be the same as we defined for scaling: 80 percent-20 percent. Historical data from the past week was used to estimate the temperature for the next day. Since there are hourly values in the data, 168 lags (24 hours * 7 days) have been selected as the input sequence length. The output sequence length was 1 and the offset was 24 hours after the last value in the input sequence. The same procedure has been replicated for the test results. These sequences have been converted into tensors. The corresponding input and output sequences were combined in a tensor data set and finally a data loader was used to construct batches 1/10 of the total size for train data. For the test data, the batch size remains the same as the overall size of the test data.

2.2 Multivariate single value forecasting:

For multivariate forecasts, we use all 12 variables in our data to produce input sequences. Note that 'TempKelvin' is present in both input and output data. Like the univariate forecast, we perform a train-test split and scale data, but this time we need to define two scalar objects: one for input data and one for output data. This is because we need the scaler object fitted to the output data to inverse transform the forecasted values.
We use the same function to generate 168 lags 3D numpy arrays for each of the 12 variables. As this problem type is also a single value forecasting problem, the output structure stays the same with a 24-hour offset. The next steps of converting to a tensor format and then to a data loader remain the same as that of univariate problem type.

2.3 Multivariate sequence to sequence forecasting:

The method of scaling and generating input sequences is the same as that of multivariate single value forecasts. The only alteration that we need to consider is the generation of output sequences. The output is a sequence of 'TempKelvin' values. Input is the sequence of 168 lags (historical data from last week) for each of the 12 variables. For the output, we're preparing to estimate the temperature value for each hour in the next 24 hours. The output sequence is a set of temperature values for the next 24 hours. The rest of the process of using

the tensors and the data loaders remain the same as for the previous two problem types.

3. Neural Network Approach

This is a time series forecasting problem. We can therefore use Feed-Forward neural networks, Recurrent neural networks and Convolutional neural networks. In addition, we will also construct supervised learning models using feed-forward problems for comparison purposes only. The method and steps taken to build these models are explained below:

3.1 Feed-Forward Supervised Learning

Supervised learning develops a function (f) which maps input x) to output (y). Mapping y = f (x:p) is defined between y and x trying to find the best value for 'p' parameters. In the case of supervised feed-forward learning, the outputs are fed back to the networks and the weights of the connections are changed to improve the effectiveness of the model.

The pre-processing of the data required for the supervised learning is different and easy compared to forecasting. Dependent variable is isolated from the other 11 variables. Two different scaler objects have been specified for independent and dependent variables. Then 80 percent-20 percent split was generated to train and evaluate without shuffling the data collection.

Next when creating a network structure, the input layer would have the same number of nodes as the number of independent variables. After the input layer, there are two more hidden layers of 32 and 8 nodes respectively. There is also an output layer with only 1 node, as we expect only 1 value. Hidden layer 1 is connected to layer 2 by a sigmoid layer in between and hidden layer 2 is also connected to the output layer by a sigmoid layer.

3.2 Feed-Forward Forecasting

3.2.1 Univariate Feed-Forward

The data for univariate forecasting has already been pre-processed. During the design of the network structure, various values of the number of hidden layers and units in those layers have been checked. After experimenting over several iterations, the final configuration of the network was determined with 2 hidden layers, 1 input layer and 1 output layer. The sizes of the hidden layers going from input to output layer are 64 and 16, respectively. The size of the input layer is equal to the number of lags in the input series i.e., 168, while the output layer has 1 unit. It takes 40 epochs for the model learning to converge towards the minimum MSE [mean squared error] value that can be achieved.

3.3 Plain Recurrent neural networks

Recurrent neural networks recall the things they learned from previous inputs when constructing a model. RNN's also apply weights to inputs such as feed-forward neural networks but adding to that there is also a 'hidden state' vector containing information from previous inputs. This hidden state is sometimes referred to as a hidden memory cell. The output of RNNs is therefore dependent on both current inputs and previous inputs. The PyTorch modules of RNNs will take 3D input tensors. There is

also no need to change the shape of the tensors like it was changed for Feed-Forward networks.

3.3.1 Univariate plain RNN

In recurrent neural networks, the sequence is the anticipated input, so the input dimension for one sequence is 1. In this case, each value in the list [1, 2, 3, 4] was checked for the number of RNN layers and 3 RNN layers provided the best results. Followed by layers of RNN, there is a fully connected layer (linear layer). After experimenting with several values, the hidden dimensions of the layer were determined to be 24. Next is the output layer, which has 1 unit, as we expect only 1 value. The model ran for 30 epochs in order to converge to the minimum possible MSE value.

3.3.2 Multivariate plain RNN

In the case of a multivariate, there are 12 input sequences, i.e., 168 lags for all 12 variables we have in the data. The input dimension of the RNN structure will therefore be 12. Three values were checked for a range of RNN layers [1, 2, 3]. It turned out that 1 RNN layer with 64 units in the next fully connected hidden layer gives the best possible results for MSE. The output dimension remains unchanged, as we still forecast only one value. The model was trained for 30 epochs.

3.4 Long Short-term Memory (LSTM) networks

LSTM is all about the cell state and its gates. The cell state serves as a "memory" for the network. The cell state may also take the knowledge from the earlier time steps to the later time steps. Every cell state in LSTM has three gates. These gates will learn the value of the information and decide to retain it or forget it during the training session. Gates are followed by a sigmoid activation feature that generates a binary output and helps to decide whether to retain or forget it.

Forget Gate outputs a value between 0 and 1 for the previous hidden state. Closer to 0 means you forget the value and closer to 1 means you move the value to the next cell state. In the input gate, the previous hidden state and current input is transferred to the sigmoid function. You also transfer the hidden state and current input to the tanh feature to help regulate the network. Then multiply the tanh output with the sigmoid output. The sigmoid output can decide which information is necessary to keep from the tanh output. The Output Gate will be used to decide the next hidden state. The output gate takes the current cell state and the new hidden state to the next cell state.

3.4.1 Univariate LSTM

Like plain RNN, the PyTorch LSTM network can take 3D input tensors. The input data is the same as the pre-processed single value forecasting data. The input dimension remains 1 as LSTM expects the input to be a sequence. The LSTM model was tuned to the number of layers, the number of units in fully connected layers and the number of epochs. The best results were obtained for 2 hidden layers, 32 linear layer units and 35 epochs.

3.4.2 Multivariate LSTM

For multivariate LSTM, the input dimensions have been modified from 1 to 12 for sequences from 12 variables. After tuning for the number of hidden layers and the number of hidden dimensions, it was decided that there will be 1 LSTM layer and 32 units as the hidden dimension. The model takes 60 epochs to converge towards the minimum possible MSE value.

3.4.3 Sequence to Sequence LSTM

Sequence to Sequence forecasting is also a multivariate forecasting problem. Here the input shape and network structure remain the same as for normal multivariate forecasting problems. But in this case, the output of the model should be a sequence of temperature values for the next 24 hours. The input dimension remains the same as 12. The number of layers is tuned to 2, the number of hidden units in the linear layer is 64, and the number of output nodes is 24. The model takes 40 epochs to converge towards the lowest possible MSE value.

3.4.4 Bidirectional LSTM

"*To overcome the limitations of a regular RNN […] we propose a bidirectional recurrent neural network (BRNN) that can be trained using all available input information in the past and future of a specific time frame*" [1]. There are two copies of the hidden layer in the bidirectional LSTM network, the first layer matches the model based on the input sequence, and the second layer copy fits the model to the inverted copy of the input sequence. The output of both layers is concatenated. *"The use of bidirectional LSTMs has the effect of allowing the LSTM to learn the problem faster"*.[4] It has been noted that bidirectional LSTM is more useful for saving time than for achieving better results compared to unidirectional LSTM.

There is not much variation in the structure of the unidirectional and bidirectional LSTM in PyTorch. This time, the fully connected layer will have twice as many units as the hidden layer of the LSTM. These are parameters for this network: 12 as input dimensions, 24 as hidden dimensions, 128 as fully connected layer dimensions and 24 as output dimensions. Also, the number of layers is 2. The model took about 40 epochs to converge towards the minimum possible MSE value.

3.5. Gated Recurrent Units (GRU) networks

GRUs are the advanced variants of plain RNNs. GRUs do not have a cell status relative to LSTMs. They're just transmitting information using the hidden state. There are only two gates in GRU: the Update Gate and the Reset Gate. The Update Gate functions the same as the LSTM network forget and input gate. While the reset gate will determine how much of the past knowledge the GRU network will forget/remember. Due to less operations, GRUs run faster than LSTM networks. And there is no definitive answer to which of the two networks is better.

3.5.1 Univariate GRU

GRUs can also take 3D tensors as input to PyTorch. Also, like RNN and LSTM, the input dimension of the network will be 1. After tuning, it was planned to have three

layers of GRU in the network. The best results were achieved for a fully connected layer of 24 units. The output dimension of the network is 1 and the model was trained for 20 epochs to converge.

3.5.2 Multivariate GRU

In the event of a multivariate forecasting problem, the input dimension will be 12 and the output dimension will be 1. The model was optimized to achieve the best results and the final parameters of the network configuration were as follows: the number of layers was 2, the size of the linear layer was 64 and it took 60 epochs to train the model.

3.5.3 Sequence to Sequence GRU

Similar to the sequence forecasting for RNN and LSTM, the GRU also has similar parameters with input (12) and output (24) measurements. After tuning, the number of layers was selected as 2 and the number of units in fully connected layers was selected as 64. The model had to run for 40epochs in order to converge towards the lowest possible MSE.

3.6 Convolutional neural networks (CNN)

CNNs are typically used when the data is 2D images, but they can also be used for 1D and 3D data. The main part of the CNN modules are the convolutional layers that perform the convolution process. Convolutions is the method of multiplying inputs with filter weights. The filter is smaller than the actual size of the data. Filters are often referred to as kernels. Typically, not all the information in the data is useful. Convolutions are used to extract the important information from data and remove the redundant data.

"*Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is. For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face*" [2].

3.6.1 Univariate CNN

CNN also takes 3D tensors as input to PyTorch. Unlike LSTM, it doesn't take the full sequence as input, though. We therefore need to describe the input dimension as 168 (hourly values of past 1 week). The network consists of three 1D convolutional layers. Convolutional layers apply filters of different sizes but identical shapes (kernel size = 1). Each convolutional layer is followed by a batch norm layer where the size of the batch norm layer is equal to the output of the previous convolutional layer. There is a pooling layer after each batch normal layer. The type of pooling is max (when a filter is applied it selects the maximum value out of all the values in that filter). It was found that the addition of a pooling layer improved the efficiency of the CNN model. The output dimensions of all convolutional layers are 125, 100 and 64 respectively. There are three linear layers accompanied by convolutional layers. The input dimensions of these fully connected layers are 64, 32 and 8. The output dimension is 1 since we only have to

estimate one temperature value. The model has been trained for 30 epochs.

3.6.2 Multivariate CNN

The multivariate CNN model also has three convolutional and three linear layers. The network structure is almost the same as the univariate CNN structure. The input dimensions of the three convolutional layers are 168, 64 and 32 respectively. Here the output of the third convolutional layer is not the same as the input of the first linear layer because there are 12 sequences for 12 variables. So, the output shape of the third convolutional layer needs to be multiplied by 12 (16 * 12) for the input value of the first linear layer. The input dimensions of successive linear layers are 32 and 16, respectively. The output dimension is 1 and it took 30 epochs to train the model.

4. Performance Evaluation and Results

Time series forecasting for temperature is a function approximation problem. Hence, MSE (mean squared error) is defined as the loss function for this project. Loss calculated on the testing data and computational efforts required to train the model, are the two measures used to compare the performance of the different models. To compare the model based on MSE loss, we let the model run until the loss graph almost becomes zero slope i.e., there is no further decrease in the MSE loss.

Minimum Sample Size:

Every model had to run for several epochs, and it took different amounts of time for the models to run. After every epoch, test MSE is stored in a numpy array. The test MSE after the last epoch is what is used to compare different models. But one MSE value is not enough for comparison as the values keep fluctuating after every iteration. In statistics, for a sample to be of large size, the minimum number of observations is 30.

"*Typically, statisticians say that a sample size of 30 is sufficient for most distributions.*".[3] Hence, the model was run for 30 iterations and the mean and standard deviation of the MSE was used to compare different models.

A table has been made for comparison which has the values for 4 parameters to compare various models. The 4 evaluating parameters are: Number of epochs, running time for 1 iteration, Mean of the MSE and Standard deviation of the MSE. The table can be seen on the next page.

Train-Test MSE loss graph:

With the MSE values on the y-axis and number of epochs on the x-axis, a graph has been plotted to observe the convergence of the learning of the model. A few examples of such graphs can be seen below:

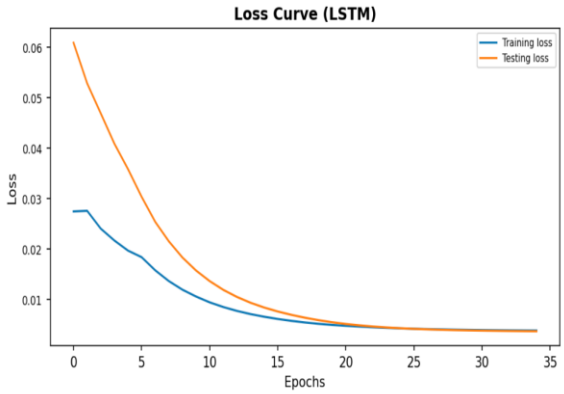

Fig 3. Train-Test MSE curve (Univariate LSTM)

			Epochs	Time taken for 1 Training	Mean MSE (30 samples)	Standard Deviation MSE (30 samples)
	Supervised Learning	Feed Forward	30	29	0.000326	0.000188
Univariate	Single Value Forecasting	Feed Forward	40	33	0.006175	0.004003
		plain RNN	30	33	0.003589	0.000055
		LSTM	35	49	0.00363	0.000071
		GRU	20	31	0.00346	0.000046
		CNN	30	27	0.003866	0.000236
Multivariate	Single Value Forecasting	plain RNN	30	39	0.003539	0.000298
		LSTM	60	87	0.003237	0.000024
		GRU	60	97	0.003143	0.000149
		CNN	30	39	0.000682	0.000539
	Sequence to Sequence	LSTM	40	108	0.00246	0.000133
		GRU	40	103	0.002258	0.000221
	Bidirectional	LSTM	40	87	0.002496	0.000262

Fig 3. Results and evaluation parameters for each model

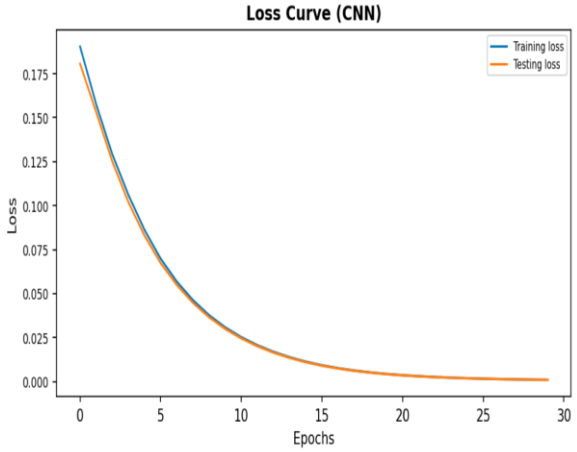

Fig 5. Train-Test MSE curve (univariate CNN)

Predictions vs Actual Values graph:

In this graph, we plot the temperature values from the test data versus the predictions made on test data. The overlapping of both the lines in the graph is a good sign of model fitting.

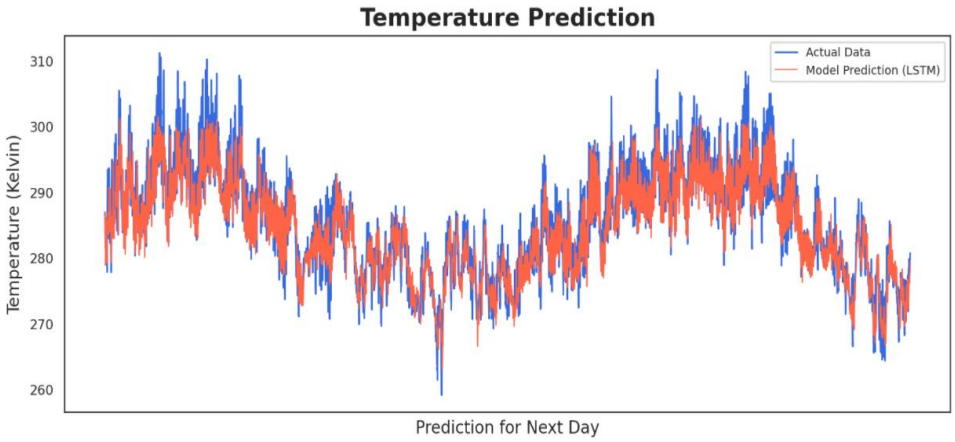

Fig 6. Actual values vs Predicted values (Seq to Seq LSTM)

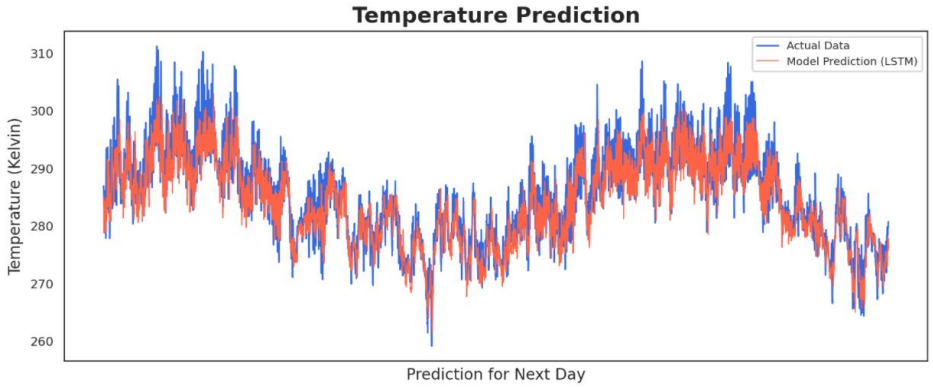

Fig 7. Actual values vs Predicted values (LSTM Seq To Sq Bidirectional)

5. Discussion

Different types of models were tested on various problems like single value forecasting, multivariate, sequence to sequence etc. Some of the models produce better results while some get trained faster than others. There is no definite answer for which one is the best model but here are few of interesting comparisons and observations:

1] Amongst all the multivariate models trained for single value forecasting, CNN is the best performing model with respect to the MSE obtained as well as the computational effort. The MSE averaged over 30 samples for CNN is 0.000682. The next best value is more than 5 times of this value. Adding to that CNN takes 30 epochs to train as compared to 60 epochs LSTM and GRU. Similarly, the time required to train the model (39 seconds) is also way smaller than the one required for LSTM and GRU (87s and 97s respectively).

CNN is performing better on a time series forecasting data than LSTM. The reason for

this might be the redundancy present in the data. There is a lot of information in the data, which is not needed, and convolutional filters help get rid of the unwanted data.

2] As mentioned earlier, bidirectional LSTM are known for training faster than producing better results. A comparison was made between bidirectional LSTM and unidirectional LSTM and GRU. It was observed that for the similar network structure, bidirectional LSTM trains in 87 seconds while it takes a little more than 100 seconds for unidirectional LSTM and GRU networks. The mean MSE obtained is almost the same with the values of 0.00246 and 0.002496 for unidirectional and bidirectional LSTM, respectively.

3] Feed forward networks, when used for supervised learning, generate a MSE of 0.000326 (STD: 0.000188). But when the feed-forward network was used with time series forecasting the MSE increased to 0.006175 (STD: 0.004003). This MSE value is nearly twice of MSE that was achieved for other networks.

4] Except for the CNN, adding more variables (multivariate) does not improve the performance of the model. For all the other networks, the mean MSE stays in the range 0.003 to 0.0035.

5.1 Future Scope

Different types of neural networks and architectures have been tried on this dataset. But as it is rightly said there is always a scope of improvement. It has been observed that CNN for multivariate dataset is the performing model for this model, while LSTM has a natural tendency to remember knowledge from the past for a long time. Hence, our plan for the future is to combine the power of CNN and memory cells of LSTM into a single architecture and try to get better results than the ones obtained now.

Secondly, this project is a short-term forecasting project. Future scope of this project involves forecasting the temperature over a longer range of values. For e.g., using daily average temperature from the past 2-3 months to predict the temperature in the longer future. These are the two things that would be the future scope of the project.

References:

[1] *Mike Schuster and Kuldip K. Paliwal, Bidirectional Recurrent Neural Networks, Published On: Nov 1997*

[2] *Page 342, Deep Learning (Adaptive Computation and Machine Learning series) Illustrated Edition, Book Published in 2016*

[3] Article Link: *https://statisticsbyjim.com/basics/central-limit-theorem/*
Uploaded on Oct 28th, 2018

[4] Article Link: *https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/*
Uploaded on June 16th, 2017

Acknowledgements:

We would like to thank our professor Dr. Chilukuri Mohan for being an excellent instructor who helped us gain all the knowledge that is required to complete this project. He also guided us through the entire project by providing complete instructions on the ways to approach this project.

Appendix:

Progress made after 1st December:

1] All the models were trained for 30 iterations and final MSE values on the test data were collected. Then by calculating the mean and standard deviation of the MSE, a comparison was made between different models addressing the same problem.

2] A final successful attempt was made to tune the multivariate sequence to sequence GRU and multivariate sequence to sequence bidirectional LSTM. Now, sequence to sequence LSTM and GRU and bidirectional LSTM have the MSE values in the same range (~0.0023).

3] The latest and updated code can be found in this GitHub repository .