VIT®

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

WEATHER PREDICTION USING LSTM

MADE BY:-

ADITYANSH SHARMA – 18BIT0342
PRIYANSHU GUPTA – 18BIT0146

ABSTRACT

Weather forecasting has become an important field of research in the last few decades. Daily Weather forecasting is used for multiple reasons in multiple areas like agriculture, energy supply, transportations etc. In this project, a neural network-based algorithm for predicting the temperature will be implemented. The Neural Networks package supports different types of training or learning algorithms. One such algorithm is Back Propagation Neural Network (BPN) technique. The main advantage of the BPN neural network method is that it can fairly approximate a large class of functions. The proposed idea is tested using the real time dataset.

Backpropagation is a method used in artificial neural networks to calculate a gradient that is needed in the calculation of the weights to be used in the network. It is commonly used to train deep neural networks a term referring to neural networks with more than one hidden layer.

INTRODUCTION

The chaotic nature of the atmosphere implies the need of massive computational power required to solve the equations that describe the atmospheric conditions. This is resulted from incomplete understanding of atmospheric processes mean that forecasts become less accurate as the difference in time between the present moment and the time for which the forecast is being made increases. Weather is a continuous, data-intensive, multidimensional, dynamic and chaotic process and these properties make weather prediction a big challenge. Generally, two methods are used for weather forecasting

(a) the empirical approach and

(b) the dynamical approach.

The first approach is based on the occurrence of analogs and is often referred by meteorologists as analog forecasting. This approach is useful for predicting local-scale weather if recorded data’s are plentiful. The second approach is based on equations and forward simulations of the atmosphere and is often referred to as computer modeling. The dynamical approach is only useful for modeling large-scale weather phenomena and may not forecast short-term weather efficiently.

OBJECTIVE

To predict the weather temperature using attributes like– humidity, precipitation, dew point and wind speed humidity etc. And using time series-data.

LITERATURE REVIEW

Similar Research work have been done previously by some of which includes

1. “”ANN Approach for Weather Prediction using Back Propagation Ch.Jyosthna Devi B.Syam Prasad Reddy,K.Vagdhan Kumar,B.Musala Reddy,N.Raja Nayak”:-

This paper include how neural networks are useful in forecasting the weather and the working of most powerful prediction algorithm called back propagation algorithm was explained . A 3-layered neural network is designed and trained with the existing dataset and obtained a relationship between the existing non-linear parameters of weather. Now the trained neural network can predict the future temperature with less error.

2. Another Research paper Classification and Prediction of Future Weather by using Back Propagation Algorithm-An Approach by Sanjay D. Sawaitul, , Prof. K. P. Wagh , Dr. P. N:-

Concludes that the new technology of wireless medium can be used for weather forecasting process The system increases the reliability, accuracy and consistency of identification and interpretation of weather images. It also concludes that the Back Propagation Algorithm can also be applied on the weather forecasting data. Neural Networks are capable of modeling a weather forecast system. The neural network signal processing approach for weather forecasting is capable of yielding good results and can be considered as an alternative to traditional meteorological approaches.

3. An Effective Weather Forecasting Using Neural Network Pooja Malik S,Prof. Saranjeet Singh Sr.   Binni Arora

This paper proposes that a new technique of weather forecasting by using Feed-forward ANN. The data is taken from Rice Research center (Kaul) Haryana. In this paper data is trained by LM algorithm. This is the fastest method among other weather forecasting methods. As there are many BP algorithm but among them Levenberg BP has better learning rate.

4. Artificial Neural Network based Weather Prediction using Back Propagation Technique

. A Multi-layered neural network is designed and trained with the existing dataset and obtained a relationship between the existing non-linear parameters of weather. The overall behavior of our model has been concluded is that by increasing the number of hidden layers, the trained neural network can classify and predict the weather variables with less error.

5. Aditya Grover, AshishKapoor and Eric Horvitz:-

in their work made a weather prediction model that predicts by considering the joint influence of key weather variables. They also made a kernel and showed that interpolation of space can be done by using GPS with such a kernel, taking into account various weather phenomena like turbulence. They also performed temporal analysis within a learner based on gradient tree and augmented the system using deep neural network.

6. Kessinger, T. R. Saxen, M. Steiner and S. Getting:-

Have shown in their work that a set of skilful predictors for thunderstorm initiation can be identified by using the random forest machine learning algorithm. The random forest method can also be used to identify "regimes" in which they can improve the skill of the application by using a forecast logic.

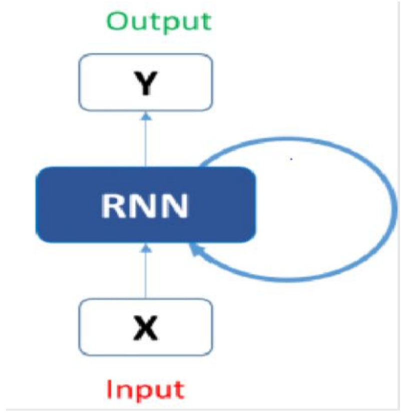
METHODOLOGY

RECURRENT NEURAL NETWORKS

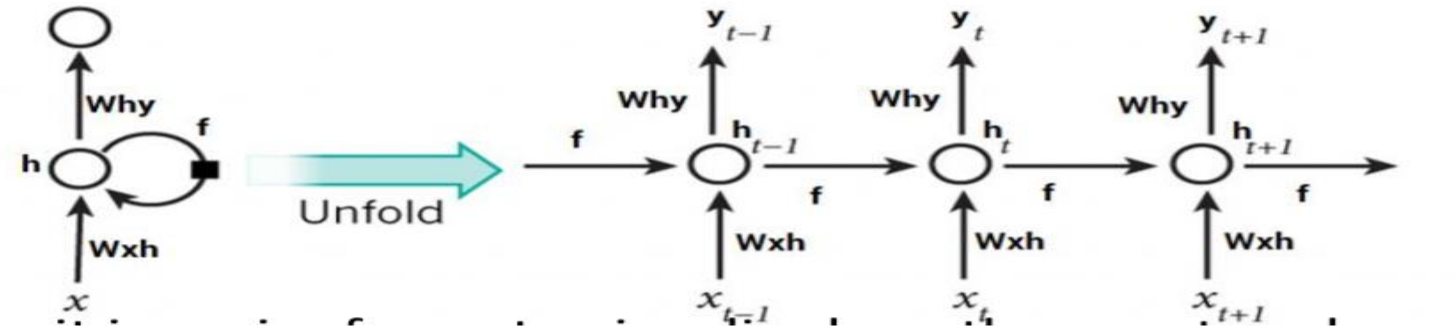
In the conventional feed-forward neural networks, all test cases are considered to be independent.

That is when fitting the model for a particular time, there is no consideration for the previous data.

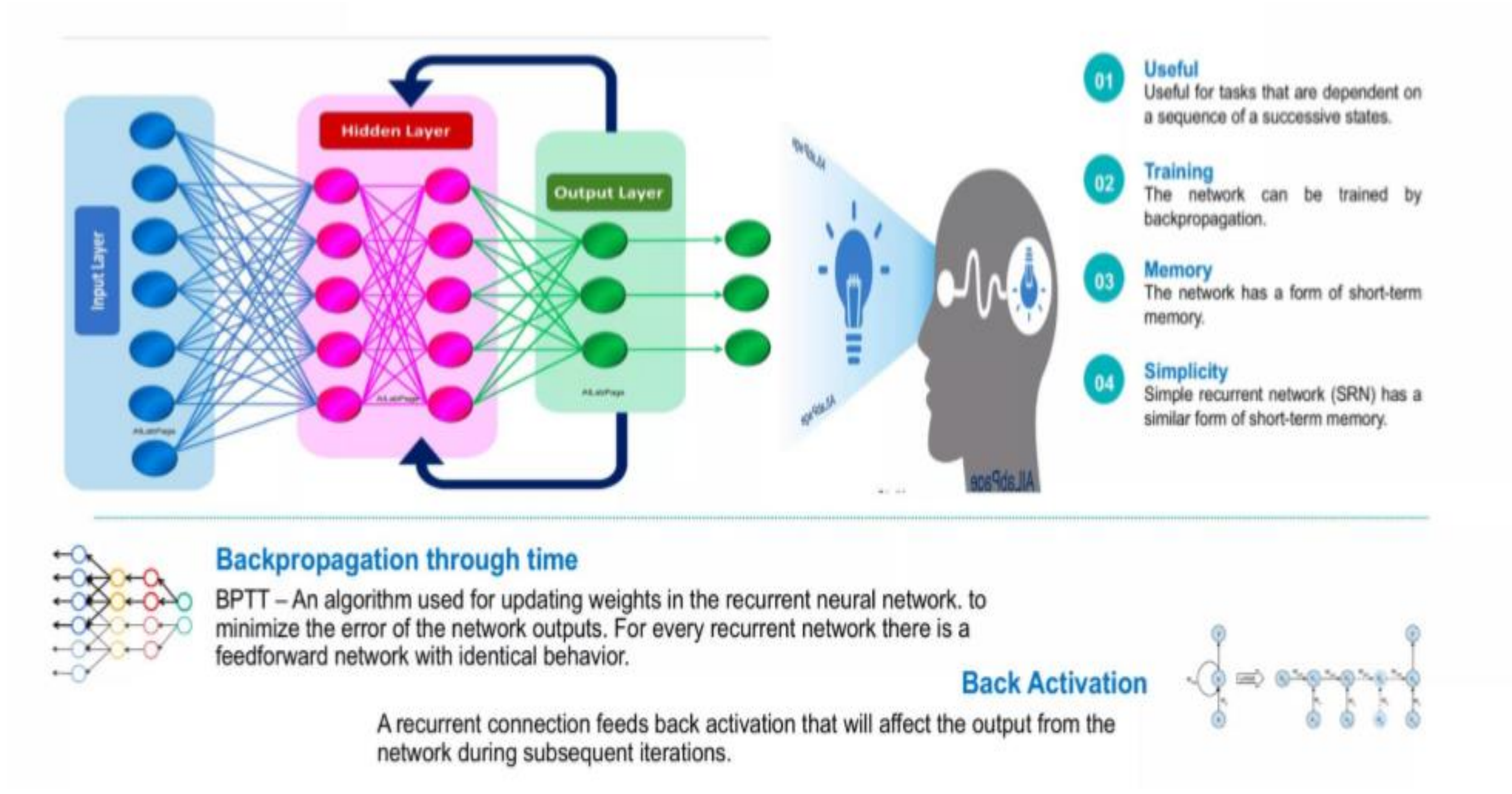
This dependency on time is achieved via Recurrent Neural Networks. A typical RNN looks like:



This may be intimidating at first sight, but once unfolded, it looks a lot simpler:



TRAINING IN RNN



Useful
Useful for tasks that are dependent on a sequence of a successive states.

Training
The network can be trained by backpropagation.

Memory
The network has a form of short-term memory.

Simplicity
Simple recurrent network (SRN) has a similar form of short-term memory.

Backpropagation through time

BPTT – An algorithm used for updating weights in the recurrent neural network. to minimize the error of the network outputs. For every recurrent network there is a feedforward network with identical behavior.

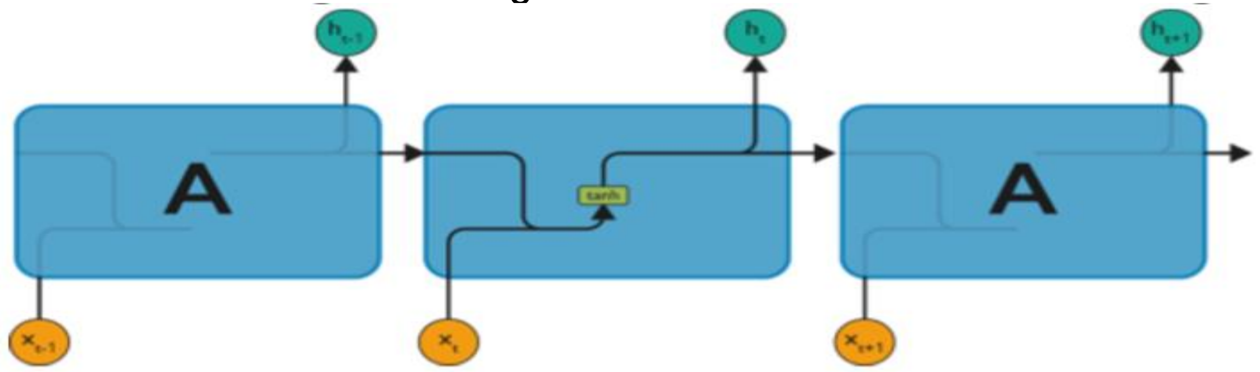
Back Activation

A recurrent connection feeds back activation that will affect the output from the network during subsequent iterations.

ARCHITECTURE OF RNN

1. Forget Gate
2. Input Gate
3. Output Gate

In a standard recurrent neural network, the repeating module consists ofone single function as shown in the below figure:



As shown above, there is a tanh function present in the layer. This function is a squashing function. So, what is a squashing function?

It is a function which is basically used in the range of -1 to +1 and to manipulate the values based on the inputs.

LIMITATION OF RNN

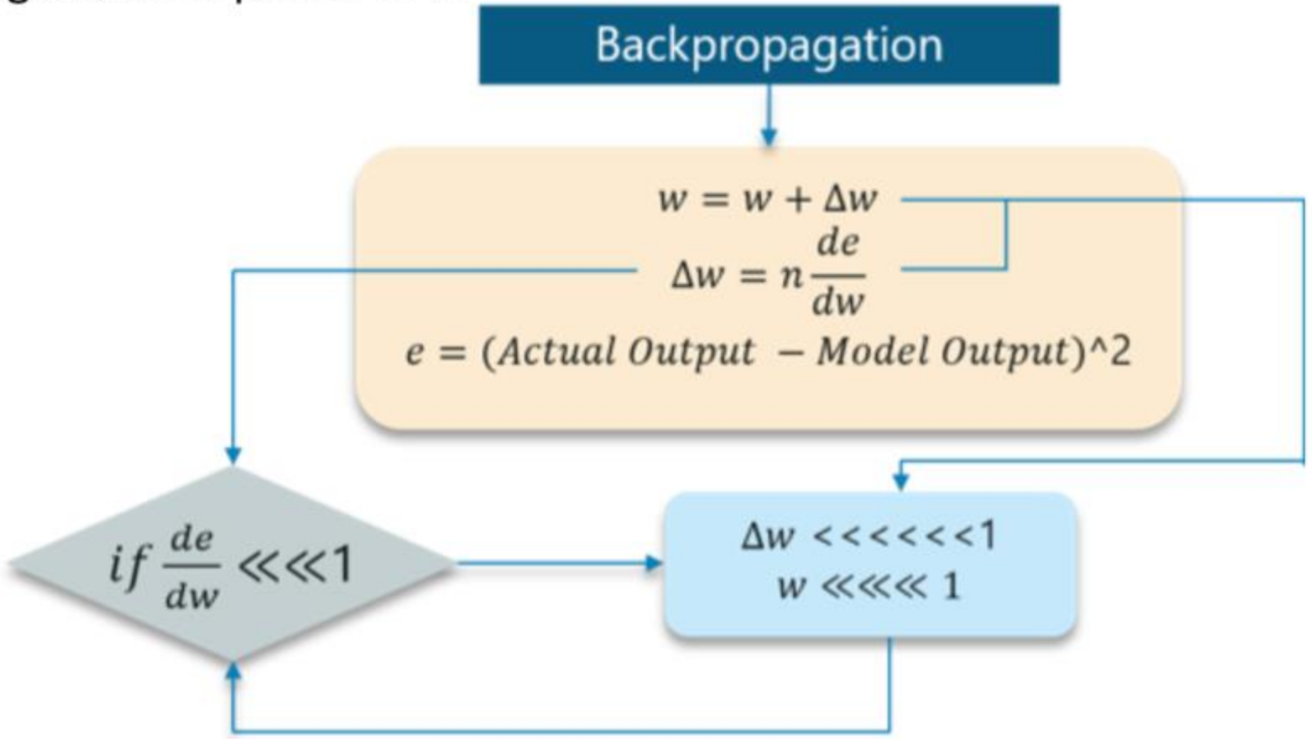
RNNs can solve our purpose of sequence handling to a great extent but not entirely.

Now RNNs are great when it comes to short contexts, but in order to be able to build a story and remember it, we need our models to be able to understand and remember the context behind the sequences, just like a human brain.

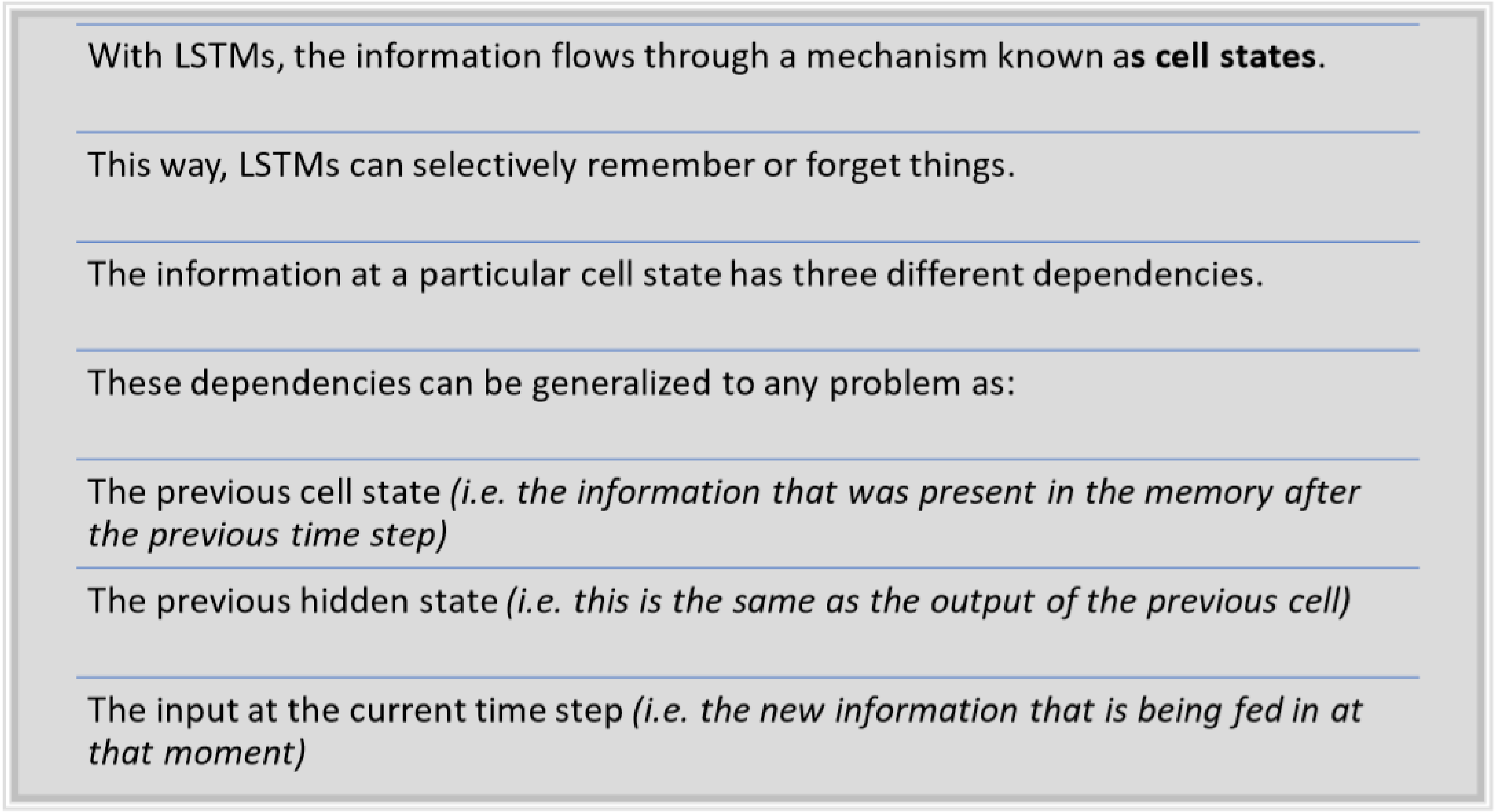
This is not possible with a simple RNN.

- The reason behind this is the problem of **Vanishing Gradient.**
- We know that for a conventional feed-forward neural network, the weight updating that is applied on a particular layer is a multiple of the learning rate, the error term from the previous layer and the input to that layer.
- Thus, the error term for a particular layer is somewhere a product of all previous layers' errors.
- When dealing with activation functions like the sigmoid function, the small values of its derivatives (occurring in the error function) gets multiplied multiple times as we move towards the starting layers.
- As a result of this, the gradient almost vanishes as we move towards the starting layers, and it becomes difficult to train these layers.
- A similar case is observed in Recurrent Neural Networks. RNN remembers things for just small durations of time, i.e. if we need the information after a small time it may be reproducible, but once a lot of words are fed in, this information gets lost somewhere.

- **Vanishing Gradient**
- When making use of back-propagation the goal is to calculate the error which is actually found out by finding out the difference between the actual output and the model output and raising that to a power of 2.


Backpropagation
$w = w + \Delta w$
$\Delta w = n\frac{de}{dw}$
$e = (Actual\ Output\ - Model\ Output)^\wedge 2$
$if \frac{de}{dw} \lll 1$
$\Delta w <<<<<< 1$
$w \lll\lll 1$

LSTM COME INTO ROLE



With LSTMs, the information flows through a mechanism known as **cell states**.

This way, LSTMs can selectively remember or forget things.

The information at a particular cell state has three different dependencies.

These dependencies can be generalized to any problem as:

The previous cell state *(i.e. the information that was present in the memory after the previous time step)*

The previous hidden state *(i.e. this is the same as the output of the previous cell)*

The input at the current time step *(i.e. the new information that is being fed in at that moment)*

VISUALIZING LSTM WITH OUR CASE OF WEATHER FORECASTING

The weather conditions of today will depend upon:

The trend that the weather has been following in the previous hours, maybe a decrease or an increase.

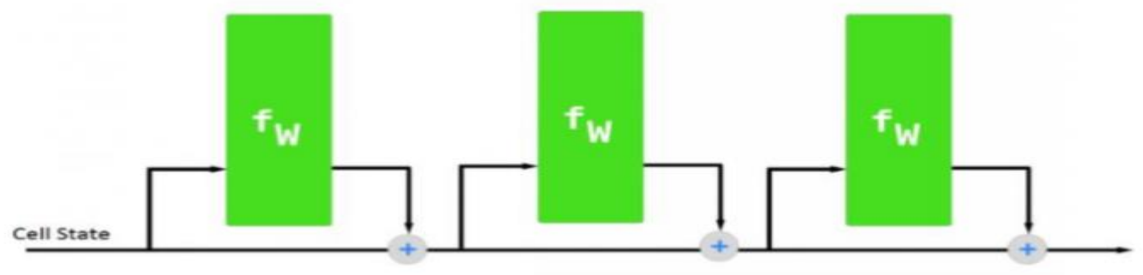
Other factors that can affect the temperature in the previous hours. These include:

PrecipitaionType, Temperature, Humidity, WindSpeed, WindBearings, Visibility,   Pressure .

Another important feature of LSTM is its analogy with conveyor belts!

We may have some addition, modification or removal of information as it flows through the different layers.

The following diagram explains the close relationship of LSTMs and conveyor belts.

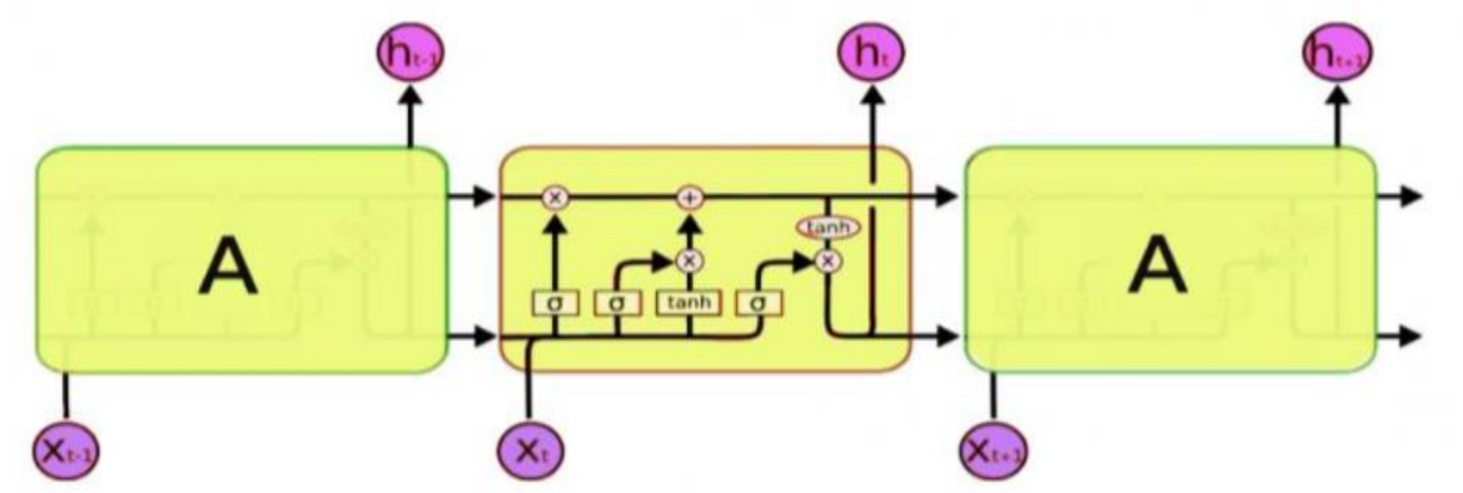


Just because of this property of LSTMs, where they do not manipulate the entire information but rather modify them slightly, they are able to forget and remember things selectively.

ARCHITECTURE OF LSTMS

A typical LSTM network is comprised of different memory blocks called cells (the rectangles that we see in the image).

There are two states that are being transferred to the next cell; the cell state and the hidden state.



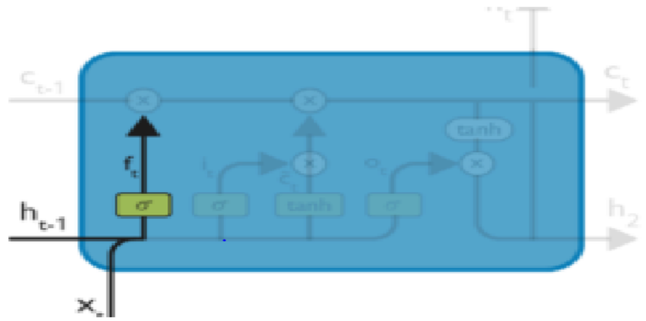
The cell state is the horizontal line in the figure and it acts like a conveyer belt carrying certain data linearly across the data channel.

A STEP-BY-STEP APPROACH TO UNDERSTAND LSTM NETWORKS BETTER

Forget gate layer

Sigmoid layer decides which data is not important

and needs to be removed.



The calculation is done by considering the new input and the previous timestamp which eventually leads to the output of a number between 0 and 1 for each number in that cell state.

As typical binary, 1 represents to keep the cell state while 0 represents to trash it.

This gate takes in two inputs; h_t-1 and x_t and then multiplied by the weight matrices and a bias is added and then the sigmoid function is applied to this value.

This vector output from the sigmoid function is multiplied to the cell state.

Input Gate

This process of adding some new information can be done via the input gate.

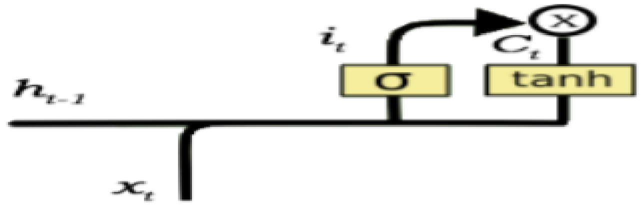
Ex: In a current state normal sunny day data is running and suddenly there is lightning and rainfall which leads to certain chenges in data that will affect the future data. So this information needs to be added in current state.

This addition of information is basically three-step process as seen from the diagram below.

Two functions –

1. Sigmoid which will again filter the information from x_t and h_t-1
2. Tanh which will create a vector containing all possible values that can be added (as perceived from h_t-1 and x_t) to the cell state.

Multiplying the value of the regulatory filter (the sigmoid gate) to the created vector (the tanh function) and then adding this useful information to the cell state via addition operation.

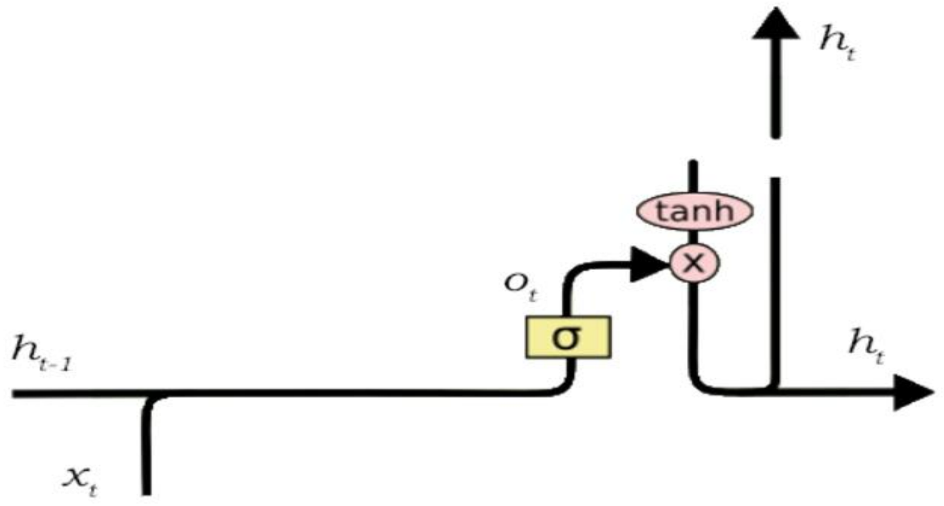


Output Gate

The job of selecting useful information from the current cell state and showing it out as an output is done via the output gate.

3 Steps:

1. Tanh function applied to cell state to get all candidate vectors that for sending as an output.
2. Sigmoid function to filter the vactors generated by tanh function.
3. Multiplying the value of this regulatory filter to the vector created in step 1, and sending it out as a output and also to the hidden state of the next cell.



IMPLEMENTATION

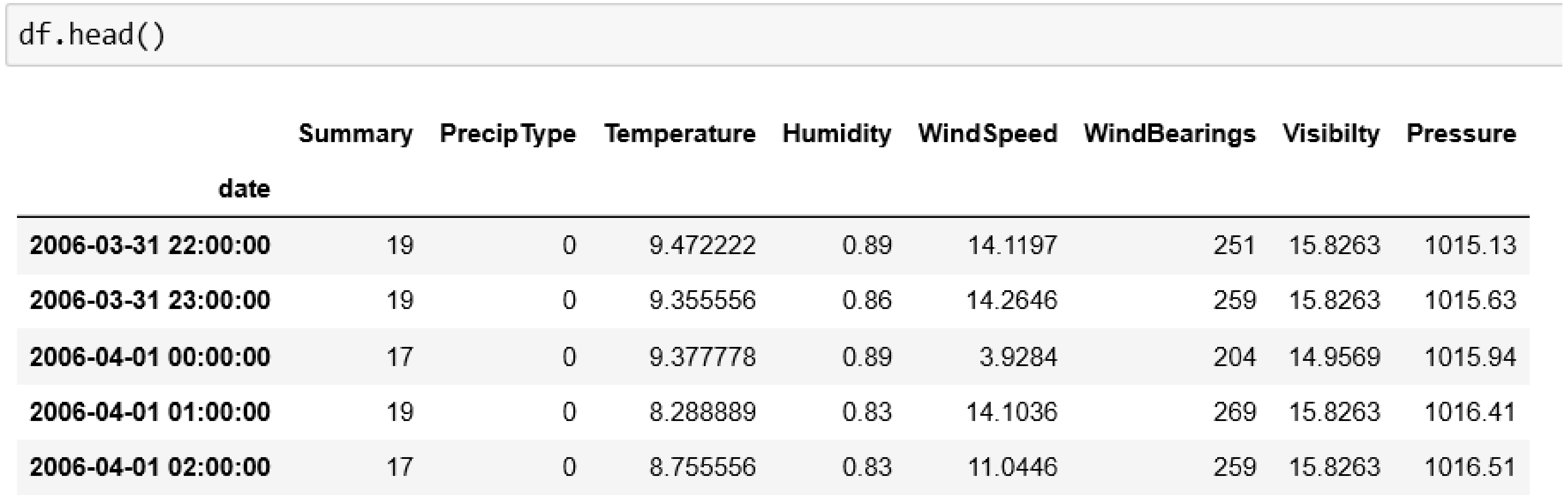
Steps-

1. Importing all necessary packages.

2. Reading the dataset. Dataset Used is obtained from Kaggle.

3. Data Visualization and Preprocessing which includes dropping unnecessary attributes, handling missing values, checking correlation and linearity, feature scaling.

After Data Preprocessing our data looks like this



```
df.head()
```

	Summary	PrecipType	Temperature	Humidity	WindSpeed	WindBearings	Visibilty	Pressure
date								
2006-03-31 22:00:00	19	0	9.472222	0.89	14.1197	251	15.8263	1015.13
2006-03-31 23:00:00	19	0	9.355556	0.86	14.2646	259	15.8263	1015.63
2006-04-01 00:00:00	17	0	9.377778	0.89	3.9284	204	14.9569	1015.94
2006-04-01 01:00:00	19	0	8.288889	0.83	14.1036	269	15.8263	1016.41
2006-04-01 02:00:00	17	0	8.755556	0.83	11.0446	259	15.8263	1016.51

4. Function to convert series to supervised learning

```
# convert series to supervised learning
def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j+1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j+1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j+1, i)) for j in range(n_vars)]
    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

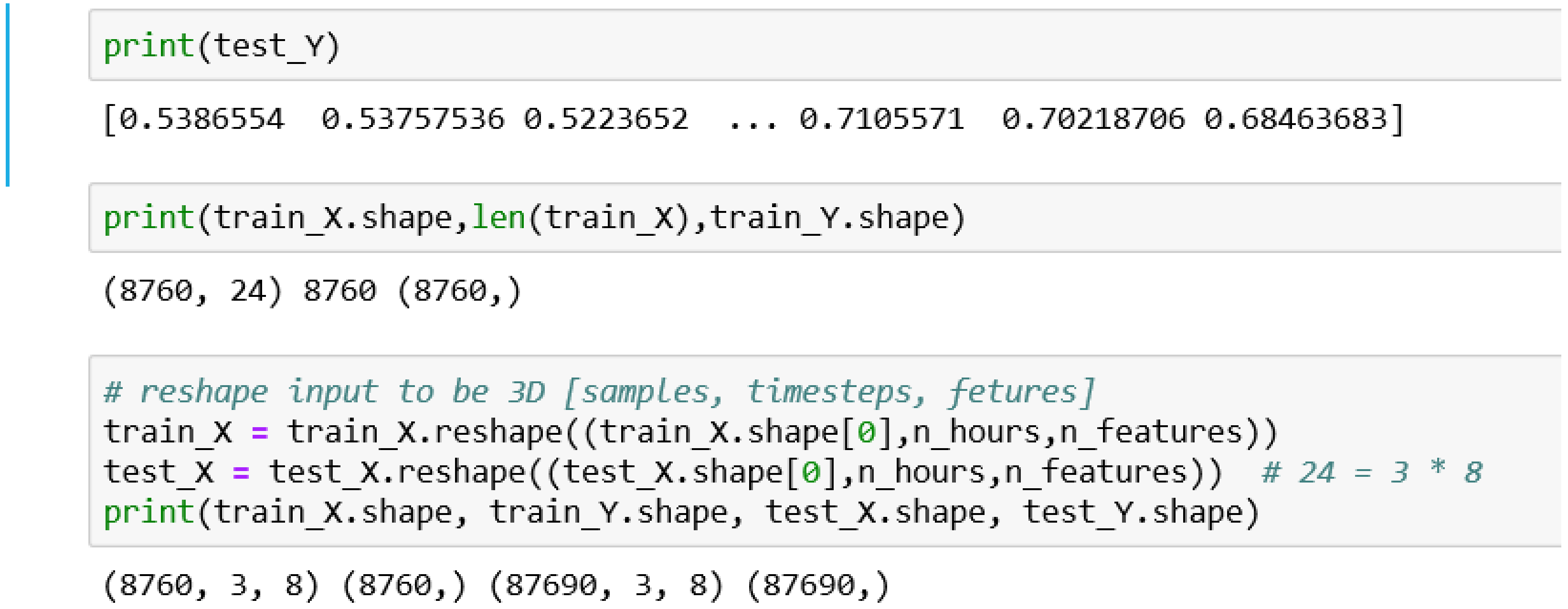
5. Train Test Data Splitting

```
n_train_hours=365*24
```

```
train = values[:n_train_hours,:]
test = values[n_train_hours:, : ]
```

```
# split into input and outputs
n_obs = n_hours * n_features
train_X,train_Y = train[:,:n_obs],train[:,-6]   # as at 6th no. from last is temp. our o/p var
test_X,test_Y = test[:,:n_obs],test[:,-6]
```

6. Reshaping input into 3D



```
print(test_Y)
```

```
[0.5386554  0.53757536 0.5223652  ... 0.7105571  0.70218706 0.68463683]
```

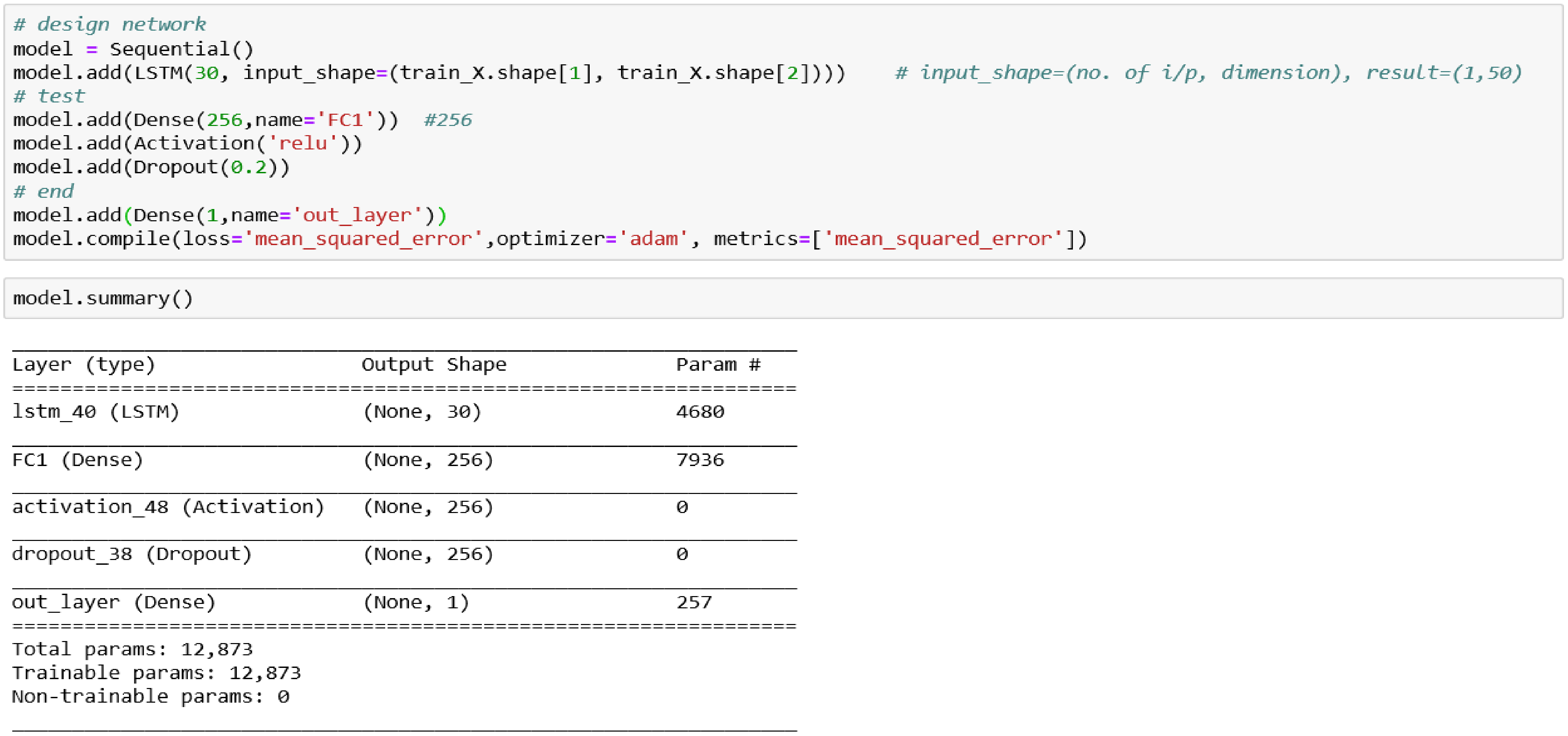
```
print(train_X.shape,len(train_X),train_Y.shape)
```

```
(8760, 24) 8760 (8760,)
```

```
# reshape input to be 3D [samples, timesteps, fetures]
train_X = train_X.reshape((train_X.shape[0],n_hours,n_features))
test_X = test_X.reshape((test_X.shape[0],n_hours,n_features))  # 24 = 3 * 8
print(train_X.shape, train_Y.shape, test_X.shape, test_Y.shape)
```

```
(8760, 3, 8) (8760,) (87690, 3, 8) (87690,)
```

7.Model Building

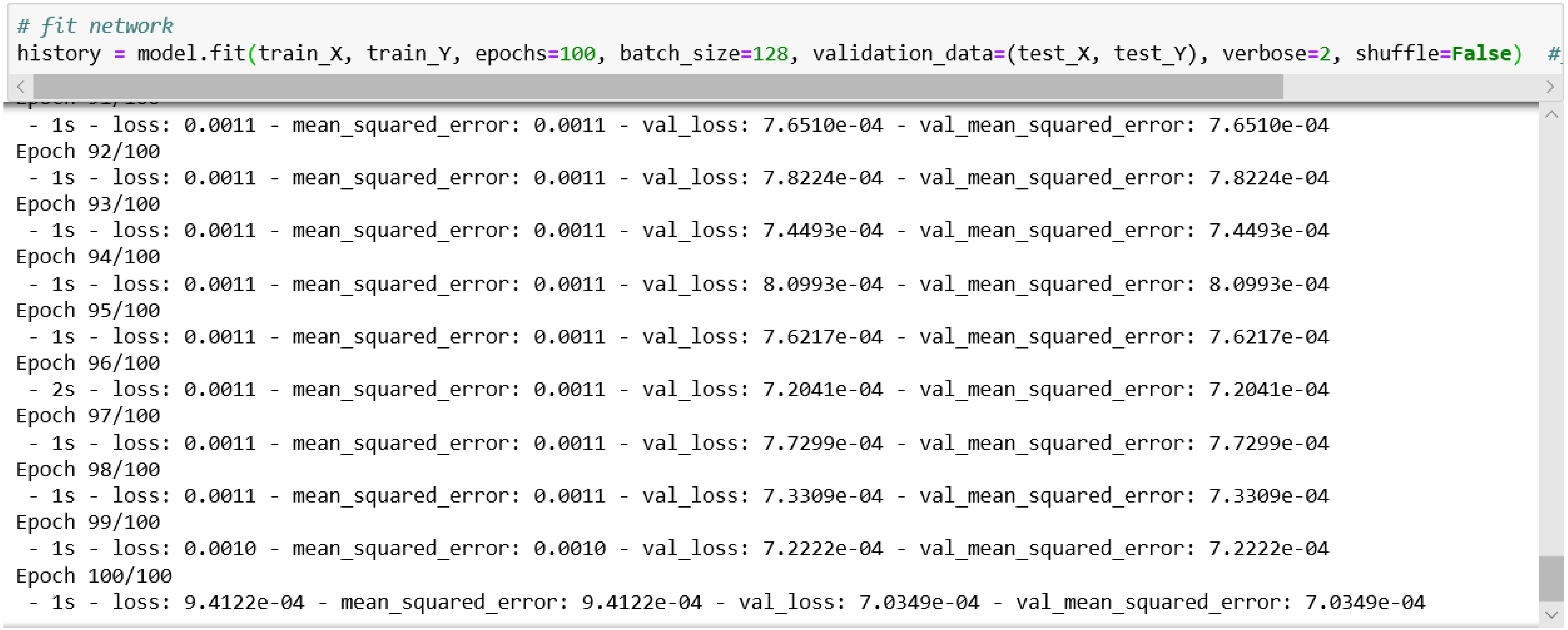


```python
# design network
model = Sequential()
model.add(LSTM(30, input_shape=(train_X.shape[1], train_X.shape[2])))    # input_shape=(no. of i/p, dimension), result=(1,50)
# test
model.add(Dense(256,name='FC1'))  #256
model.add(Activation('relu'))
model.add(Dropout(0.2))
# end
model.add(Dense(1,name='out_layer'))
model.compile(loss='mean_squared_error',optimizer='adam', metrics=['mean_squared_error'])
```

```python
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
lstm_40 (LSTM)               (None, 30)                4680      
_________________________________________________________________
FC1 (Dense)                  (None, 256)               7936      
_________________________________________________________________
activation_48 (Activation)   (None, 256)               0         
_________________________________________________________________
dropout_38 (Dropout)         (None, 256)               0         
_________________________________________________________________
out_layer (Dense)            (None, 1)                 257       
=================================================================
Total params: 12,873
Trainable params: 12,873
Non-trainable params: 0
_________________________________________________________________
```

8. Fit Model

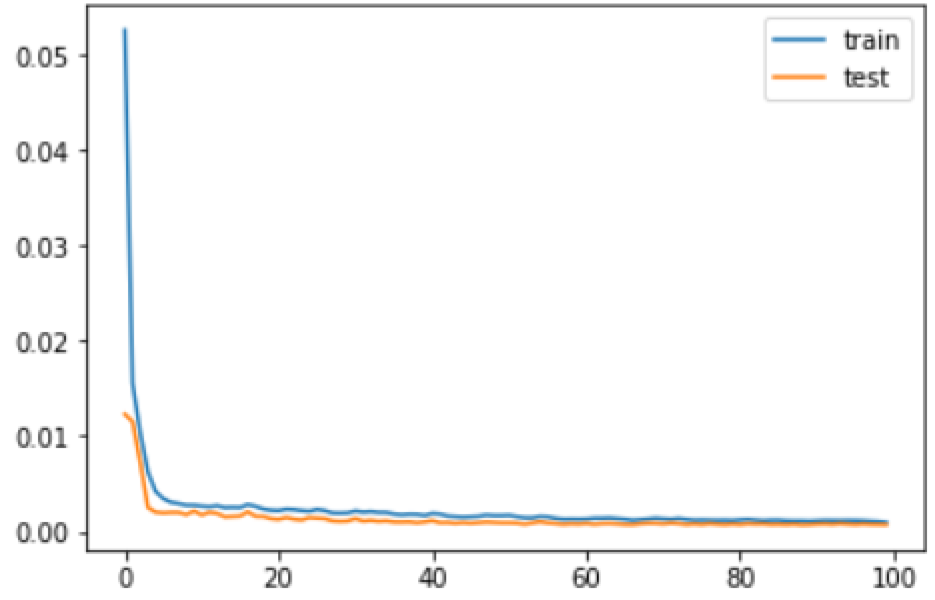


```
# fit network
history = model.fit(train_X, train_Y, epochs=100, batch_size=128, validation_data=(test_X, test_Y), verbose=2, shuffle=False)  #
```

```
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.6510e-04 - val_mean_squared_error: 7.6510e-04
Epoch 92/100
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.8224e-04 - val_mean_squared_error: 7.8224e-04
Epoch 93/100
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.4493e-04 - val_mean_squared_error: 7.4493e-04
Epoch 94/100
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 8.0993e-04 - val_mean_squared_error: 8.0993e-04
Epoch 95/100
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.6217e-04 - val_mean_squared_error: 7.6217e-04
Epoch 96/100
 - 2s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.2041e-04 - val_mean_squared_error: 7.2041e-04
Epoch 97/100
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.7299e-04 - val_mean_squared_error: 7.7299e-04
Epoch 98/100
 - 1s - loss: 0.0011 - mean_squared_error: 0.0011 - val_loss: 7.3309e-04 - val_mean_squared_error: 7.3309e-04
Epoch 99/100
 - 1s - loss: 0.0010 - mean_squared_error: 0.0010 - val_loss: 7.2222e-04 - val_mean_squared_error: 7.2222e-04
Epoch 100/100
 - 1s - loss: 9.4122e-04 - mean_squared_error: 9.4122e-04 - val_loss: 7.0349e-04 - val_mean_squared_error: 7.0349e-04
```

9. Plotting Loss and Error during training and testing

```
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='test')
pyplot.legend()
pyplot.show()
```



10. Predicting

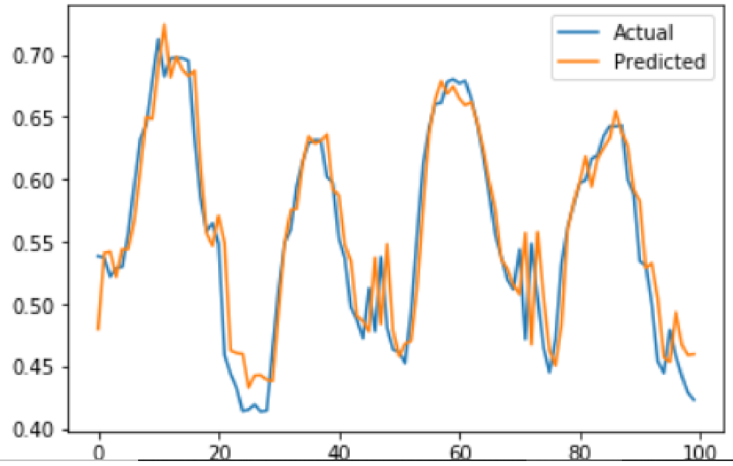
```python
# make a prediction
test_Y_predicted = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_hours*n_features))
```

```python
print(test_Y_predicted)
print(test_Y)
```

```
[[0.48036286]
 [0.541106  ]
 [0.5421792 ]
 ...
 [0.71163416]
 [0.6750506 ]
 [0.6736934 ]]
[0.5386554  0.53757536 0.5223652  ... 0.7105571  0.70218706 0.68463683]
```

```python
pyplot.plot(test_Y[:100],label="Actual")
pyplot.plot(test_Y_predicted[:100],label="Predicted")
pyplot.legend()
```

```
<matplotlib.legend.Legend at 0x17a2dbfb9e8>
```



11. Invert Scaling Back to original values from Rescaled Values

```
# invert scaling for forecast
vir_array= np.concatenate((test_X[:,-8:-6],test_Y_predicted),axis=1)
inv_test_Y_predicted = np.concatenate((vir_array,test_X[:,-5:]),axis=1)
inv_test_Y_predicted = scaler.inverse_transform(inv_test_Y_predicted)
inv_test_Y_predicted = inv_test_Y_predicted[:,2]
print(inv_test_Y_predicted)
```

```
[ 7.356138 11.398759 11.525701 ... 22.633926 20.340778 20.008066]
```

```
# invert scaling for actual
test_Y = test_Y.reshape((len(test_Y), 1))
vir_array2= np.concatenate((test_X[:,-8:-6],test_Y),axis=1)
inv_test_Y = np.concatenate((vir_array2,test_X[:,-5:]),axis=1)
inv_test_Y = scaler.inverse_transform(inv_test_Y)
inv_test_Y = inv_test_Y[:,2]
print(inv_test_Y)
```

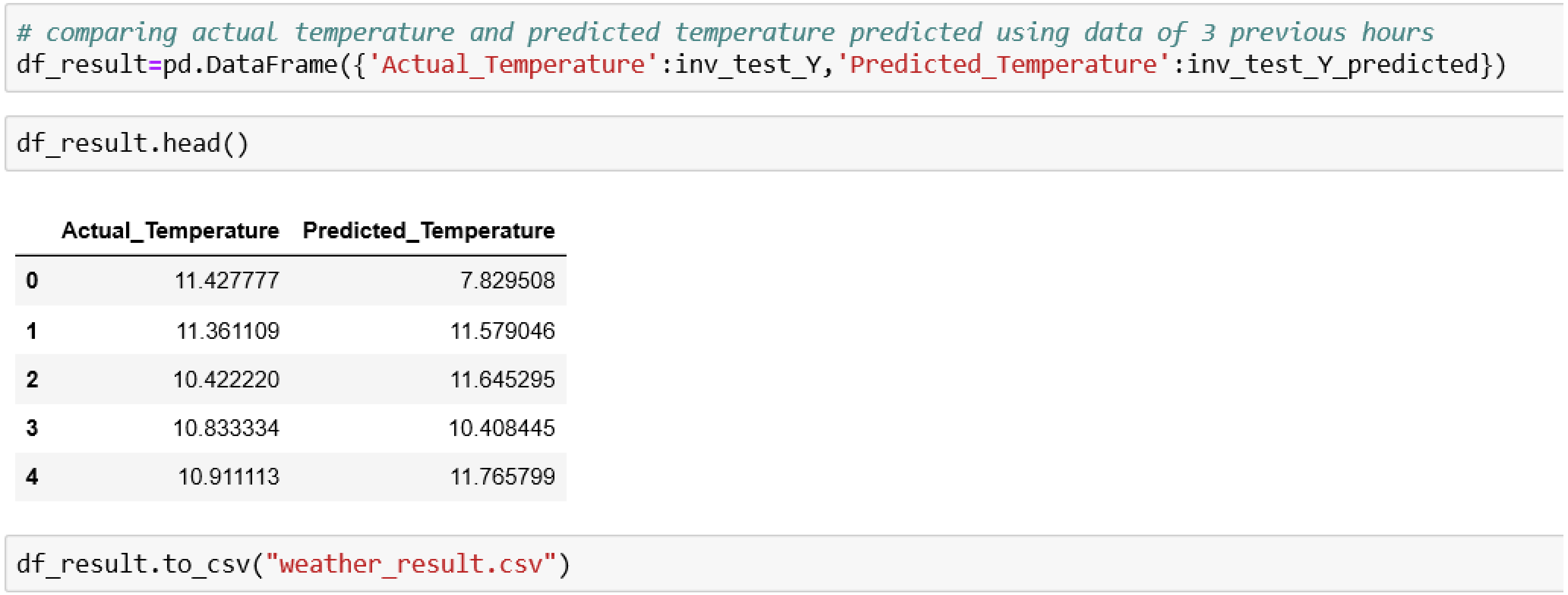
```
[11.427777 11.361109 10.42222  ... 22.038887 21.522223 20.438887]
```

12. Error Analysis

```
# calculate RMSE
from math import sqrt
rmse = sqrt(mean_squared_error(inv_test_Y, inv_test_Y_predicted))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 1.637
```

Lets see what we have have get in a table format



```
# comparing actual temperature and predicted temperature predicted using data of 3 previous hours
df_result=pd.DataFrame({'Actual_Temperature':inv_test_Y,'Predicted_Temperature':inv_test_Y_predicted})
```

```
df_result.head()
```

	Actual_Temperature	Predicted_Temperature
0	11.427777	7.829508
1	11.361109	11.579046
2	10.422220	11.645295
3	10.833334	10.408445
4	10.911113	11.765799

```
df_result.to_csv("weather_result.csv")
```

RESULTS AND DISCUSSION

Long short-term memory network the Back Propagation Algorithm is implemented and the variations in parameters are observed. According to these variations the logic in Back Propagation will be developed and the change in other parameters with respect to one parameter will be predicted.

All the information of parameters which changed and predicted is collected together and the new classification of weather will be drawn.

CONCLUSION

It concludes that there can be a new method to predict the future weather with the help of back propagation training algorithm. It was found that the network learns very fast with back propagation algorithm. The results are more accurate for predicting the future weather.

Back propagation is a gradient descent algorithm which learns by minimizing

The error in the output by adjusting the weights in the network. Our model has potential to capture the complex relationships between many factors that contribute to certain weather condition.

FUTURE SCOPE

The accuracy from weather forecasting model using LSTM and Back Propagation Algorithm is more than other Statistical Model. An extension to this technology can be done using any of the other technique instead of Data mining and Different Algorithm.

Furthermore, in order to improve the efficiency of the neural network algorithms other statistical based feature selection techniques, statistical indicators can be integrated. In another perspective fuzzy techniques can be incorporated, which an inferential, probality-based approach to data comparisons is allowing to infer, based on probabilities, the strength of the relationships between attributes in the data sets and to achieve better predictability rate.