

Weather forecasting with Recurrent Neural Networks in Python



Rohan Kosandal

Follow



Dec 29, 2019 · 5 min read



Source: Photo by [Pixabay](#) from [Pexels](#)

With advances in Data Science and Technology, we are able to get precise forecasts of the weather in almost every location around the world. Data collected from weather stations and satellites are used to analyze and predict the meteorological disasters caused by extreme weather. According to research, based on observations of the weather in the past we can predict the weather in the future. Machine Learning can be used to forecast weather with historical weather data. Remember that the weather predicted is an educated guess!

In this article, we will develop a deep learning model with Recurrent Neural Networks to provide 4 days forecast of the temperature of a location by considering 30 days of historical temperature data.

So let us begin with a brief introduction to Recurrent neural networks.

What is Recurrent neural network(RNN)?

RNN is a deep learning model that is used for Time-series prediction, speech recognition, etc. Unlike traditional neural networks, recurrent networks use their memory(also called states) to predict sequence outputs. In simple words, RNN is used when we want to predict a future outcome based on the previous sequential inputs. For example, we can use RNN to predict the next word in a sentence by providing previous words.

RNN for Weather forecasting.

Now that we have an insight about RNN so let us begin to develop an RNN model that can provide 4 days forecast of temperature based on 30 days of historical temperature data. I have used [Google colab](#) to implement this code and Spyder for visualizations, there are a lot many tools that you can use according to your preference.

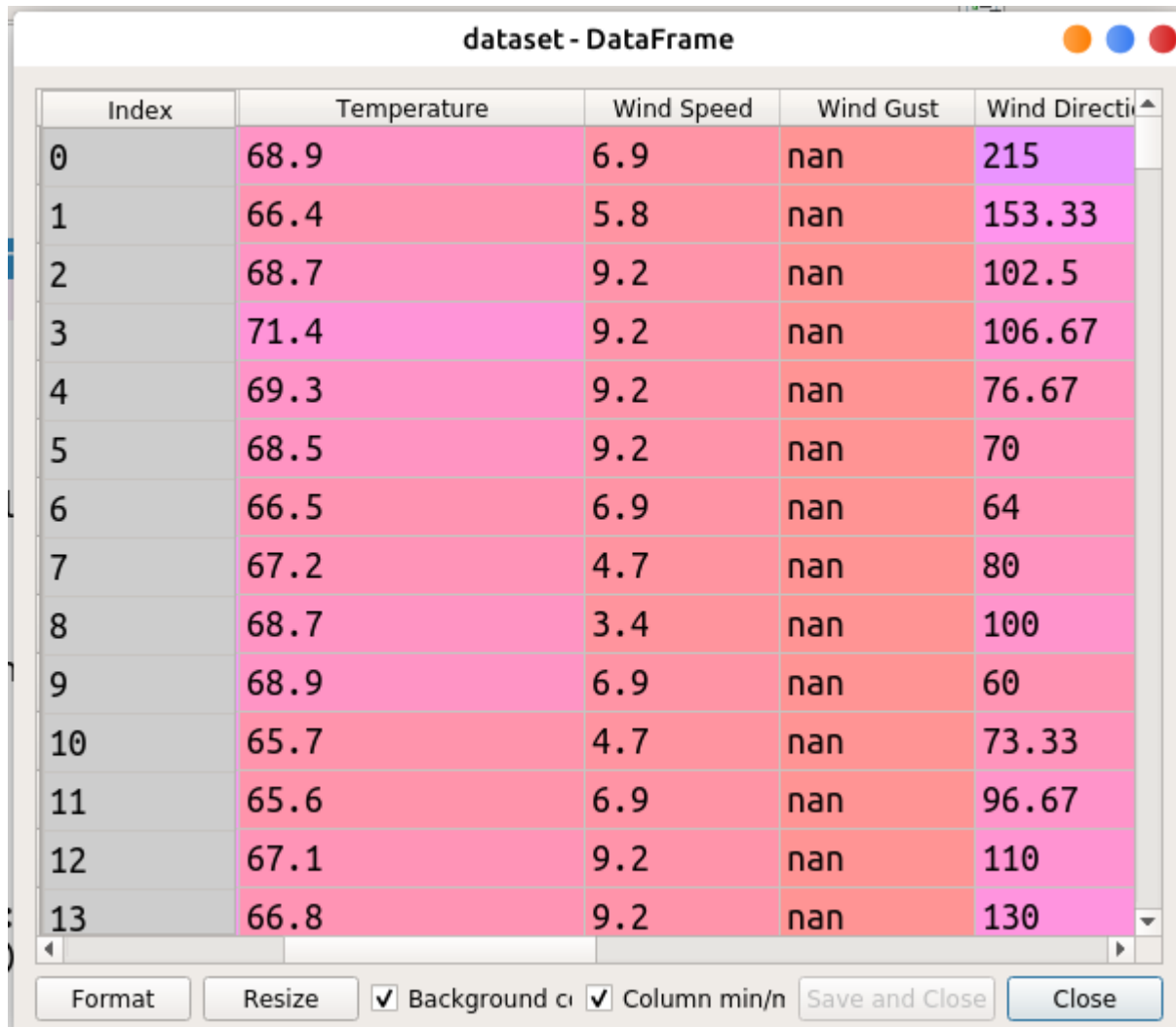
You can download historical weather dataset from [here](#), also feel free to use any weather dataset of your choice which has temperature data.

Let's load the dataset and see the first few rows:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
#import dataset from data.csv file
dataset = pd.read_csv('data.csv')
dataset = dataset.dropna(subset=["Temperature"])
dataset=dataset.reset_index(drop=True)

training_set = dataset.iloc[:,4:5].values
```



Index	Temperature	Wind Speed	Wind Gust	Wind Direction
0	68.9	6.9	nan	215
1	66.4	5.8	nan	153.33
2	68.7	9.2	nan	102.5
3	71.4	9.2	nan	106.67
4	69.3	9.2	nan	76.67
5	68.5	9.2	nan	70
6	66.5	6.9	nan	64
7	67.2	4.7	nan	80
8	68.7	3.4	nan	100
9	68.9	6.9	nan	60
10	65.7	4.7	nan	73.33
11	65.6	6.9	nan	96.67
12	67.1	9.2	nan	110
13	66.8	9.2	nan	130

We include only the temperature column as we are going to forecast temperature and drop all the rows that have no values or has a NaN.

Next, we will have to apply feature scaling to normalize temperature in the range 0 to 1.

```
#Feature Scaling
from sklearn.preprocessing import MinMaxScaler

sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(training_set)
```

We will create a training set such that for every 30 days we will provide the next 4 days temperature as output. In other words, input for our RNN would be 30 days temperature data and the output would be 4 days forecast of temperature.

```
x_train = []
y_train = []

n_future = 4 # next 4 days temperature forecast
n_past = 30 # Past 30 days

for i in range(0, len(training_set_scaled) - n_past - n_future + 1):
    x_train.append(training_set_scaled[i : i + n_past , 0])
    y_train.append(training_set_scaled[i + n_past : i + n_past +
n_future , 0 ])

x_train , y_train = np.array(x_train), np.array(y_train)

x_train = np.reshape(x_train, (x_train.shape[0] , x_train.shape[1],
1) )
```

x_train contains 30 previous temperature inputs before that day and **y_train** contains 4 days temperature outputs after that day. Since x_train and y_train are lists we will have to convert them to numpy array to fit training set to our model.

Now we are ready with our training data so let's proceed to build an RNN model for forecasting weather.

1. First, we will import keras sequential model from keras.models and keras layers ie. LSTM, Dense and dropout. You can refer Keras documentation for more info on Keras models and layers [here](#)

```
from keras.models import Sequential
from keras.layers import LSTM, Dense , Dropout
# Fitting RNN to training set using Keras Callbacks. Read Keras
callbacks docs for more info.
```

2. Let us define the layers in our RNN. We will create a sequential model by adding layers sequentially using **sequential()**. The first layer is a Bidirectional LSTM with 30 memory units, **return_sequence=True** means that the last output in the output sequence is returned and the **input_shape** describes the structure of the input. With Bidirectional LSTM the output layer gets feedback from past(forward) as well as future(backward) states simultaneously. We add 3 hidden layers and an output layer with a linear activation function that outputs 4 days temperature. And at the last, we fit the RNN model with our training data.

```
regressor = Sequential()

regressor.add(Bidirectional(LSTM(units=30, return_sequences=True,
input_shape = (x_train.shape[1],1) ) ))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units= 30 , return_sequences=True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units= 30 , return_sequences=True))
regressor.add(Dropout(0.2))

regressor.add(LSTM(units= 30))
regressor.add(Dropout(0.2))
regressor.add(Dense(units = n_future,activation='linear'))

regressor.compile(optimizer='adam',
loss='mean_squared_error',metrics=['acc'])
regressor.fit(x_train, y_train, epochs=500,batch_size=32 )
```

Note: I have used Adam optimizer because it is computationally efficient.

3. Create test data to test our model performance.

```
# read test dataset
testdataset = pd.read_csv('data (12).csv')
#get only the temperature column
testdataset = testdataset.iloc[:30,3:4].values

real_temperature = pd.read_csv('data (12).csv')
real_temperature = real_temperature.iloc[30:,3:4].values
```

```
testing = sc.transform(testdataset)
testing = np.array(testing)
testing = np.reshape(testing, (testing.shape[1], testing.shape[0], 1))
```

4. Now that we have our test data ready, we can test our RNN model.

```
predicted_temperature = regressor.predict(testing)

predicted_temperature = sc.inverse_transform(predicted_temperature)

predicted_temperature = np.reshape(predicted_temperature,
                                   (predicted_temperature.shape[1], predicted_temperature.shape[0]))
```

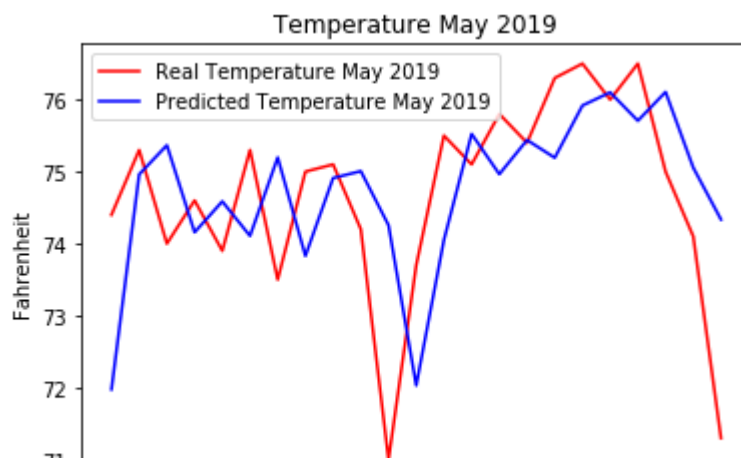
The output from the model is in the normalized form, so to get the actual temperature values we apply **inverse_transform()** to the **predicted_temperature** and then reshape it.

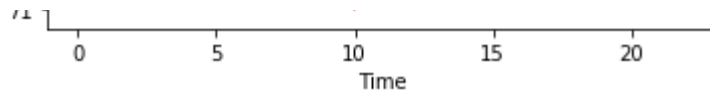
Let's compare the predicted and real temperatures. As we can see the model performs well with the given test data.

```
real_temperature
array([[82.],          [82.],          [83.],          [83.]])

predicted_temperature
array([[83.76233 ], [83.957565], [83.70461 ], [83.6326  ]])
```

If we forecast temperature for a month and visualize it we get the following results.





Forecast of temperature over a month

Conclusion

Recurrent neural networks are the best known for time-series predictions as they can process sequence data and also they can be integrated with convolutional neural networks (CNN) for processing sophisticated hybrid models, but also they do take up a lot of memory for processing. I would like to suggest some of the ways to you can follow to increase the accuracy of the model:

1. Increase or decrease the number of epochs. Try with a different number of epochs like 100, 200, 300... and so on
2. One of the best ways is to use a large dataset and train the model, but this might take a longer time. You can use GPUs with Keras by installing CUDA. Refer to [this](#) article for more information.
3. Add more LSTM layers to the RNN.

Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! [Take a look.](#)

Get this newsletter

Emails will be sent to nicocarb@stanford.edu.
[Not you?](#)

Machine Learning

Recurrent Neural Network

Weather Prediction

Deep Learning

Long Short Term Memory

[About](#) [Write](#) [Help](#) [Legal](#)

Get the Medium app

