Capstone Project: Air Passenger Forecasting

Ananth Sethuraman

1 Rationale for Air Passenger Forecasting

Airlines, airport operators, taxi operators and hotels are interested in air passenger forecasting. As an example, let there be a forecast that the number of air passengers at Oakland Airport will double in five years. The forecast will probably have the following consequences:

- Airlines will probably have to schedule a greater number of flights to, and from, Oakland Airport
- Oakland Airport will probably have to hire additional staff.
- Taxi operators will probably have to dispatch a greater number of taxis to Oakland Airport.
- Hotels near Oakland Airport will probably have to hire additional staff.

2 Techniques of Air Passenger Forecasting

Air passenger forecasting is a subset of a field known as time series analysis. Historically,time series analysis has used techniques such as exponential moving average and ARIMA. After the advent of machine learning, recurrent neural networks (RNNs) have become important as well.

In this project, we will discuss how an RNN can be used to solve an air passenger forecasting problem.

3 The Dataset

We will pick up a dataset titled *International Airline Passengers: Monthly Totals January 1949—December 1960* from [2, p531]. The dataset looks like

this:

Month	Air Passengers
Jan 1949	112,000
Feb 1949	118,000
Mar 1949	132,000
Apr 1949	129,000
May 1949	121,000
...	
Dec 1949	118,000
Jan 1950	126,000
...	
Mar 1952	193,000
...	
Nov 1960	390,000
Dec 1960	432,000

The first row is saying that there were 112,000 air passengers in January 1949; the second row, that there were 118,000 air passengers in February 1949; the third row, that there were 132,000 air passengers in March 1949; and so on. The last row is saying that there were 432,000 air passengers in December 1960.

Why pick this dataset? The reason is given in Appendix A.

4 The Forecasting Problem

4.1 A Preliminary Attempt to State the Forecasting Problem

Let us imagine ourselves to be in January 1949; we want to forecast how many air passengers there will be in January 1952, 36 months from now. (Of course that number is in the dataset, but we will not peek at the row that houses the number—we want to compute the number.)

After this let us imagine ourselves to be in February 1949; we want to forecast how many air passengers there will be in February 1952, 36 months from now.

Then let us imagine ourselves to be in March 1949; we want to forecast how many air passengers there will be in March 1952, 36 months from now.

And so on.

We will use the foregoing as a preliminary version of the problem we want to

solve. As the word "preliminary" suggests, we will embellish the problem with further details, nuances, and refinements in the sections that follow.

4.2 Forecast Horizon

The phrase "36 months from now" occurred repeatedly in the discussion above. The number 36 is termed the forecast horizon. The forecast horizon is usually an adjustable parameter or an input parameter. That is to say, our software should be written in such a way that the user can change the forecast horizon to 24 months, 17 months, 31 months, or, indeed, any positive integer number of months.

For example, let the foreacast horizon be 17 months. Then the discussion above (Section 4.1) could be reworded in the following way:

> Let us imagine ourselves to be in January 1949; we want to forecast how many air passengers there will be in June 1952, 17 months from now.
>
> After this let us imagine ourselves to be in February 1949; we want to forecast how many air passengers there will be in July 1952, 17 months from now.
>
> Then let us imagine ourselves to be in March 1949; we want to forecast how many air passengers there will be in August 1952, 17 months from now.

A family of forecasting problems can be generated in this way by letting the forecast horzon be an adjustable parameter or an input parameter.

4.3 Mathematical Notation

Let t be a month; $t+1$, the month after t; $t+2$, the month after $t+1$; and so on. In this notation, $t+36$ is the month that comes 36 months after t. Further, let $t-1$ be the month before t; $t-2$, the month before $t-1$; and so on.

For example, let t = Mar 1949; then $t+1$ = Apr 1949; $t+2$ = May 1949; $t+36$ = Mar 1952; $t-1$ = Feb 1949; $t-2$ = Jan 1949.

Further let $P(t)$ denote the number of air passengers that is listed in the dataset for the month t.

For example, let t = March 1949. Then $P(t) = P(\text{Mar 1949}) = 132{,}000$; $P(t+1) = P(\text{Apr 1949}) = 129{,}000$; $P(t+2) = P(\text{May 1949}) = 121{,}000$; $P(t) = P(\text{Mar}$

1952) = 193,000. $P(t-1) = P$(Feb 1949) = 118,000; $P(t-2) = P$(Jan 1949) = 112,000;

We will now introduce a notation for forecasts. Let us imagine ourselves to be in a month t; let s be a month that comes later than t. Let us generate a forecast for what the number of air passengers in s; denote the forecast as $P_f(t, s)$; the subscript f is short for "forecast."

For example, let t = Mar 1949 and s = Mar 1952. Let us imagine ourselves to be in Mar 1949, and let us generate the forecast that there will be 200,000 air passengers in Mar 1952. Then

$$P_f(t, s) = P_f(\text{Mar 1949}, \text{Mar 1952}) = 200,000$$

Note that $P_f(t, s)$ is a function of two variables t, s, with $s > t$. The difference $s - t$ is nothing but the forecast horizon.

4.4 Restate the Forecasting Problem with Mathematical Notation

Now that we have a mathematical notation, we can make another attempt to state the forecasting problem:

Let h be a positive integer, and let its value be known *a priori*. We will term h the forecast horizon. Let t denote a month, and let it vary over the set

$$\{\text{Jan 1949, Feb 1949, Mar 1949, } \ldots, \text{ Dec 1957}\}$$

The forecasting problem is to compute $P_f(t, t + h)$

For example, let t = Mar 1949 and $h = 36$. Let us imagine that we are in Mar 1949, and let us generate the forecast that there will be 200,000 air passengers 36 months from now. Then

$$P_f(t, t+h) = P_f(\text{Mar 1949}, \text{Mar 1949+36}) = P_f(\text{Mar 1949}, \text{Mar 1952}) = 200,000$$

4.5 Add Training and Testing Phases to the Forecasting Problem

Thus far, we have not specified what the training and testing subsets are. Let us specify the training subset to be Jan 1949 through Dec 1956, and the testing subset to be from Jan 1957 through Dec 1957. The forecasting problem can be divided in two as follows.

In the training phase, let

$$t \in \{\text{Jan 1949, Feb 1949, Mar 1949, } \ldots, \text{ Dec 1956}\}$$

Compute iterates of $P_f(t, t+h)$, and use these iterates in order to compute the loss function or objective function in the training phase.

In the testing phase, let

$$t \in \{\text{Jan 1957, Feb 1957, Mar 1957, } \ldots, \text{ Dec 1957}\}$$

Compute $P_f(t, t+h)$, and quote these computations as the forecasts. Use these forecasts to compute an error metric.

Note that the phrase "iterates of $P_f(t, t+h)$" is used in the training phase, but not in the testing phase. That is because the weights are considered to be adjustable quantities in the training phase, and to be non-adjustable (or frozen) quantities in the testing phase.

Note also that the phrase "loss function or objective function" is used in the training phase, but not in the testing phase.

4.6 Recurrent Nature of the Forecasting Problem

The use of an RNN to solve a problem is justified if the problem lends itself to the word "recurrent." Let us go into that now.

We stipulate that $P_f(t, t+h)$ should be computed in a definite sequence. Let us say, as an example, that $h = 36$. We stipulate that $P_f(\text{Jan 1949}, \text{Jan 1952})$ be computed first, $P_f(\text{Feb 1949}, \text{Feb 1952})$ be computed second, $P_f(\text{Mar 1949}, \text{Mar 1952})$ be computed third, and so on.

We will allow $P_f(\text{Jan 1949}, \text{Jan 1952})$ to be used as an input to computing $P_f(\text{Feb 1949}, \text{Feb 1952})$. Likewise, we will allow $P_f(\text{Jan 1949}, \text{Jan 1952})$ and $P_f(\text{Feb 1949}, \text{Feb 1952})$ to be used as inputs to computing $P_f(\text{Mar 1949}, \text{Mar 1952})$

The general rule is that we will allow

$$P_f(t-1, t+35), P_f(t-2, t+34), P_f(t-3, t+33), \ldots$$

to be used as inputs to computing $P_f(t, t+36)$.

It is this general rule that makes the problem lend itself to the word "recurrent'." Compare the general rule with concepts like recursion, recursive formula, which are used both in computer science and in mathematics.

The numbers

$$P_f(t-1, t+35), P_f(t-2, t+34), P_f(t-3, t+33), \ldots$$

will be termed the recurrent inputs in the computation of $P_f(t, t+36)$

In practice, we will also place an upper bound on the number of recurrent inputs. Suppose, for example, the upper bound is 5. It means that

$$P_f(t-1, t+35), P_f(t-2, t+34), P_f(t-3, t+33), P_f(t-4, t+32), P_f(t-5, t+31)$$

will be the recurrent inputs in the computation of $P_f(t, t+36)$ Here is another way of saying the same thing: $P_f(t-6, t+30)$ is considered to be too old a forecast to be used as an input in computing $P_f(t, t+36)$.

4.7 Dependence on Available Exact Values

We will also allow certain values of $P(t)$ to be used as input to the computation of $P_f(t, t+h)$. More specifically, we will allow

$$P(t), P(t-1), P(t-2), \ldots$$

to be used as inputs to the computation of $P_f(t, t+h)$ These inputs are termed external inputs, in order to distinguish them from the recurrent inputs.

In practice we will place an upper bound on the number of external inputs. Suppose, for example, that the upper bound is 4. It means that

$$P(t), P(t-1), P(t-2), P(t-3)$$

can be used as inputs to the computation of $P_f(t, t+h)$. Here is another way of saying the same thing: $P(t-4)$ is considered to be too old a value to be used as input to the computation of $P_f(t, t+h)$.

5 Exploratory Data Analysis

5.1 Is the Dataset Clean?

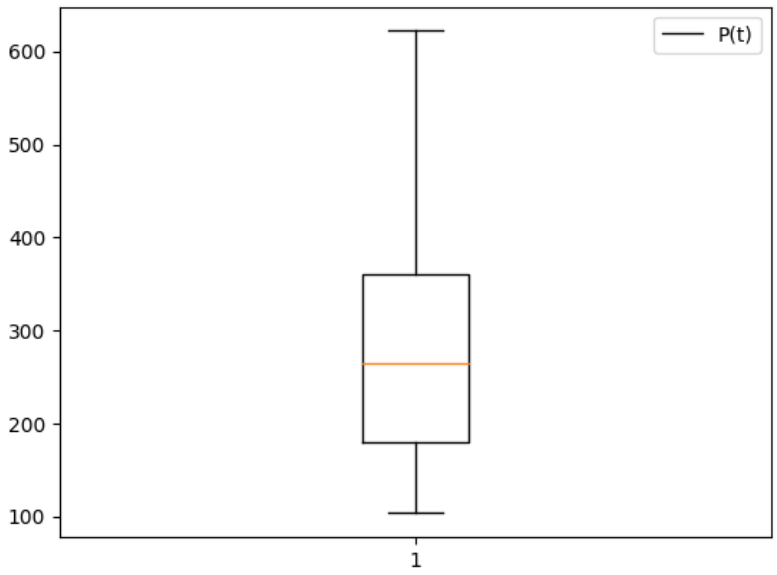
There are no missing entries, no nans, in the dataset.

5.2 Five Number Summary

The five-number summary of $P(t)$ is:

minimum	104,000
25th percentile or Q_1	180,000
50th percentile or median	265,500
75th percentile or Q_3	360,500
maximum	622,000

The five-number summary is depicted in the boxplot below:



5.3 Any Outliers?

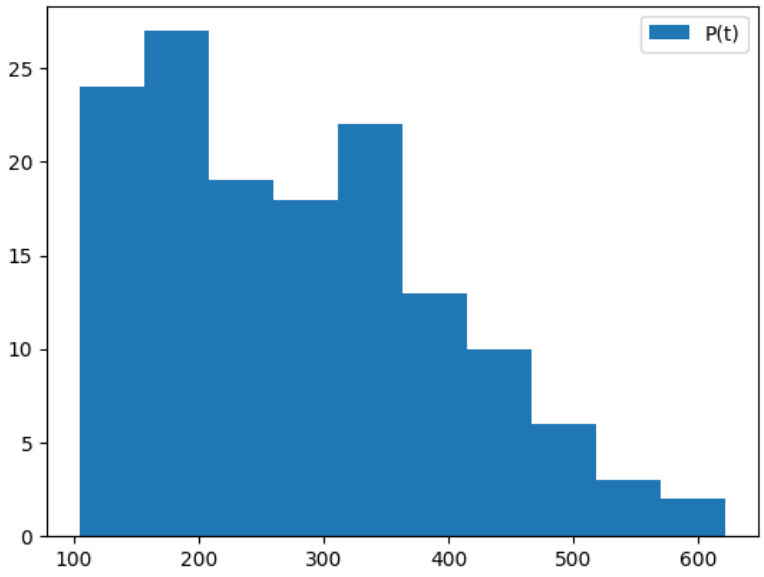
Observe that

$$Q_1 - 1.5(Q_3 - Q_1) = 180,000 - 1.5(360,000 - 180,000) \quad = -90.000$$
$$Q_3 + 1.5(Q_3 - Q_1) = 360,000 + 1.5(360,000 - 180,000) \quad = 630,000$$

All values of $P(t)$ are included in the interval $-90,000 \ldots 630,000$, so there are no outliers.

5.4 Histogram of $P(t)$

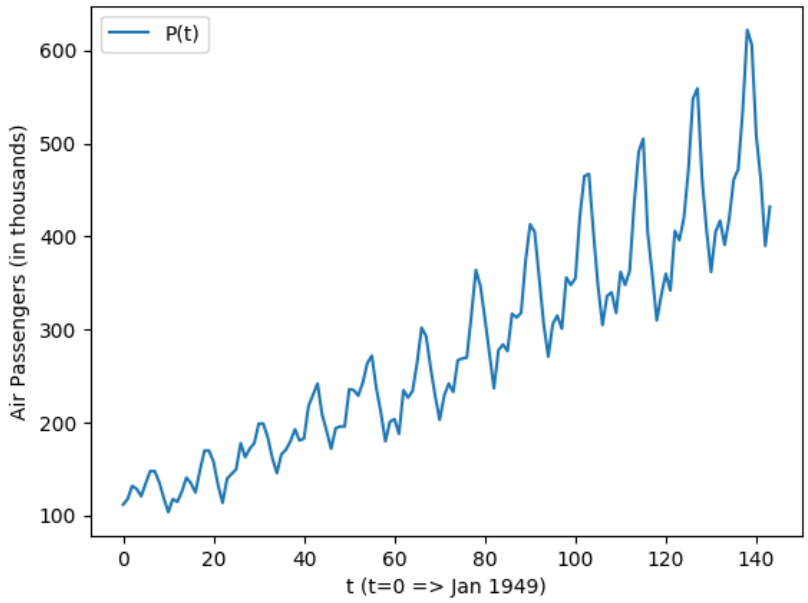
Here is the histogram of $P(t)$ with 10 bins



The shape of the histogram is very far removed from the normal (or bell-shaped) histogram, suggesting that there is a systematic trend in $P(t)$.

5.5 Linear and Seasonal Trends in $P(t)$

Here is a graph of $P(t)$ as a function of t.



The graph of $P(t)$ as a function of t shows a linear trend and a seasonal trend.

Linear Trend

To spot the linear trend, mark the local minimums of $P(t)$. Join the local minimums by a smooth curve. The smooth curve is roughly a straight line, and can be termed the linear trend.

Seasonal Trend

Imagine that the linear trend were to be subtracted from $P(t)$. What would remain of $P(t)$ would still show an oscillatory behavior. The time period of the oscillations is 12 months. It seems that there is a local maximum in late July or early August each year. There is also local maximum in December each year. The public travels a lot in these months. There is a local minimum in November each year; perhaps the public uses November to make purchases for Christmas, and therefore minimize travel in November. The oscillatory behavior can be termed the seasonal trend.

If ARIMA were to be used to solve the forecasting problem, we would have to isolate the linear and seasonal trends, by means of special formulas or algorithms [2]. But do we have to do this in machine learning also? I suggest that the answer is *No,* it is not obligatory to isolate the linear and seasonal trends.

Machine learning should have the ability to, well, learn that the dataset has a linear and a seasonal trends. In my code, I have not attempted to isolate the linear and seasonal trends.

Not all in the machine language community would agree with my approach. Brownlee [3] does isolate the linear and seasonal trends. More on this topic in Section 7.4.

6 Description of the Code

6.1 Read in the Dataset

The dataset is available in the form of a `.csv` file in some websites. The `.csv` file can be downloaded, and read in by means of the standard Python routine `pandas.read_csv`. The routine `pandas.read_csv` will house the dataset as a `dataframe` instance. Denote the `dataframe` instance as `D_orig`. `D_orig` will have two columns. One column will house the various values of t, *i.e.,*

$$\text{Jan 1949, Feb 1949, Mar 1949, } \ldots \text{, Dec 1960}$$

The other column will house the values of $P(t)$.

It will be convenient to house the second column of `D_orig` in a `numpy` array named `P`.

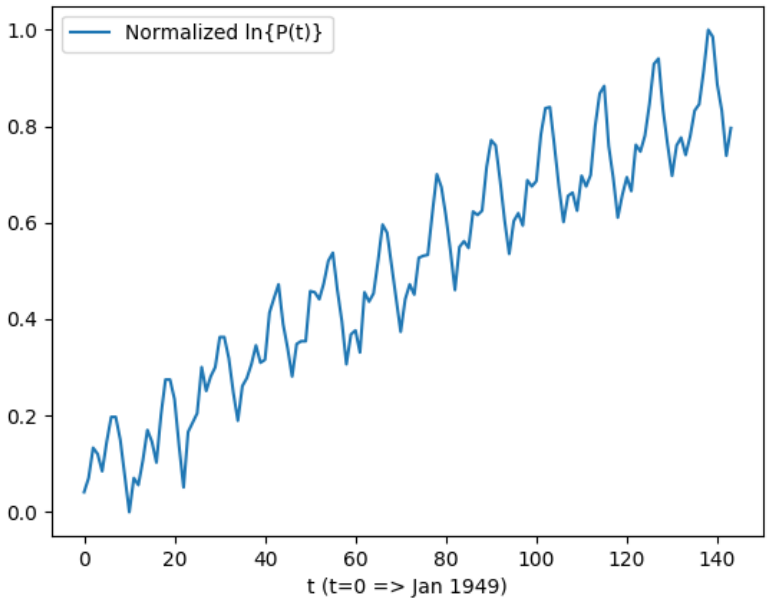
6.2 Keeping $P(t)$ to a Bounded Range

The minimum of $P(t) = 102$, the maximum 622. It seems a good idea to reduce the range of $P(t)$ by two moves:

First, compute the natural log of $P(t)$ by means of the standard routine `numpy.log`.

Second, normalize $\ln P(t)$ to the interval $[0, 1]$ by means of the routines `fit`, `transform` and `inverse_transform` that are associated with `sklearn.preprocessing.MinMaxScaler`.

In the code I use the notation `norma_ln_P`. The graph of this quantity as a function of t looks like this:



6.3 Training and Testing Sets

We will use the values of $P(t), t =$ Jan 1949 . . . Dec 1956 as the the `train_x` array. In the language of `numpy` array slices, this would be the subarray `P[0:96]`. (Recall from Section 6.1 that `P` is a `numpy` array that houses the values of $P(t)$.)

We will use the values of $P(t), t =$ Jan 1957 . . . Dec 1957 as the `test_x` array. In the language of `numpy` array slices, this would be the subarray `P[96:132]`.

Let us use a variable `FORECAST_HORIZON` to house, well, the forecast horizon. Suppose, as an example, the forecast horizon is 36. The `train_y` is determined as the slice of $P(t)$ that is 36 months ahead of `train_x`, *i.e.,* `P[0+36:96+36]`.

Likewise, the `test_y` is determined as the slice of $P(t)$ that is 36 months ahead of `test_x`, *i.e.,* `P[96+36:132+36]`.

6.4 Reshaping `train_x, train_y, test_x, test_y`

As mentioned above (Section 6.3) `train_x, train_y, test_x, test_y` are one-dimensional `numpy` arrays. The tutorial examples of the `keras LSTM` API sug-

gest that they be reshaped into three-dimensional arrays.

The third dimension of the three-dimensional arrays is 1. This is because $P(t)$ is the number of passengers, a real number. That is to say, $P(t)$ is a function that maps t to $\mathbb{R}^1$. If $P(t)$ were to map t to, say, $\mathbb{R}^7$, then the third dimension of the three-dimensional arrays would be 7.

For the second dimension of the three-dimensional arrays, my code uses a parameter named `NUM_TIME_POINTS_PER_SUB_VECTOR`. I suggest that the parameter should be at least as big as the upper bound which was mentioned in Section 4.7. Suppose, for example, that `NUM_TIME_POINTS_PER_SUB_VECTOR` $= 3$. I suggest that it means that the RNN will think that values like $P(t-4), P(t-5), \ldots$ are too old to be used in computing $P(t)$. This is just my impression from reading the `keras` documentation.

For the first dimension of the three-dimensional arrays, my code uses these formulas:

```
first dimension of train_x = len (train_x) / NUM_TIME_POINTS_PER_SUB_VECTOR
first dimension of train_y = len (train_y) / NUM_TIME_POINTS_PER_SUB_VECTOR
first dimension of test_x  = len (test_x ) / NUM_TIME_POINTS_PER_SUB_VECTOR
first dimension of test_y  = len (test_y ) / NUM_TIME_POINTS_PER_SUB_VECTOR
```

6.5 Layers

My code uses three layers: (i) a `keras LSTM` layer; (ii) a `keras Dense` layer, together with a `TimeDistributed` wrapper; and (iii) a `keras Activation` layer. I chose these after seeing some tutorial examples on `keras's LSTM` API.

`keras LSTM` Layer

My code creates a `keras LSTM` layer in this way:

```
keras.layers.recurrent.LSTM (units = NUM_HIDDEN_NEURONS,
                             input_shape = (NUM_TIME_POINTS_PER_SUB_VECTOR, 1),
                             return_sequences = True)
```

The quantity `NUM_HIDDEN_NEURONS` is a parameter. I suggest that this quantity be greater than, or equal to, the upper bound that was mentioned in Section 4.6.

The argument `input_shape` is a pair that comprises the second and third dimensions of the reshaped arrays Section 6.4.

`keras Dense` Layer

My code creates a `keras Dense` layer in this way:

```
keras.layers.core.Dense (units = 1)
```

The argument `units = 1` is determined by the third dimension of the reshaped arrays (Section 6.4).

`keras Activation` Layer

My code creates a `keras Activation` layer in this way:

```
keras.layers.Activation (ACTIVATION)
```

where `ACTIVATION` is a parameter.

6.6 Error Metric

A number of error metrics have been proposed in literature, and have been used in software competitions of time series forecasts. Examples are MAPE, MdAPE, sMAPE, sMdAPE, MdRAE, and GMRAE. Hyndman & Koehler [4, p3] have reviewed these formulas and find that "many of these proposed measures of forecast accuracy are not generally applicable, can be infinite or undefined, and can produce misleading results." Therefore Hyndman & Koehler [4, p12] present an error metric named MASE, short for Mean Absolute Scaled Error, that is free of these issues.

My code uses MASE.

Please see Appendix B for some details, including an example how to compute MASE.

Let us note a few key points about MASE:

- Barring one exception, MASE is always a non-negative real number. That one exception is the case that the time series is a constant function of time. In that exceptional case, MASE is undefined.
- The smaller the value of MASE, the more accurate the forecast.

7 Results

First, let us go over the adjustable parameters in my code. The adjustable parameters are:

```
FORECAST_HORIZON
NUM_HIDDEN_NEURONS
NUM_TIME_POINTS_PER_SUB_VECTOR
ACTIVATION
```

I experimented with these values:

```
FORECAST_HORIZON = 36
NUM_HIDDEN_NEURONS = 16, 24, 32, 40, 48, 64, 128,256
NUM_TIME_POINTS_PER_SUB_VECTOR = 6
ACTIVATION = 'relu'
```

7.1 MASE Values

Recall that MASE is being used as an error metric (Section 6.6). With the `keras` LSTM model, the MASE values were as follows:

# Hidden Neurons	MASE
16	4.02
24	4.02
32	0.74
40	0.67
48	0.58
56	0.52
64	0.50
128	0.43
256	0.37
512	4.02

I suggest that if the number of hidden neurons is less than 32 there is underfitting. Likewise, I suggest that, if the number of hidden neurons is greater than 256 there is overfitting.

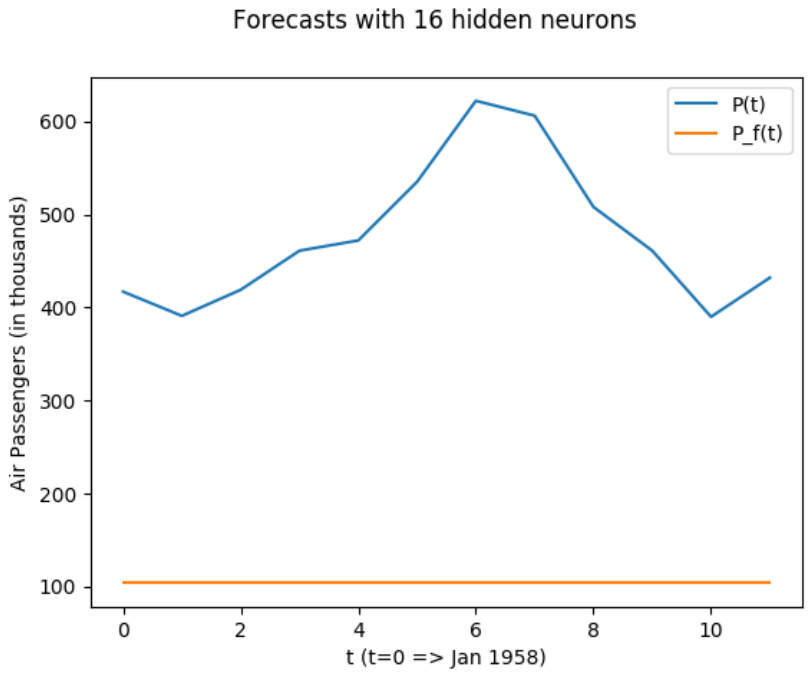
ARIMA as a benchmark

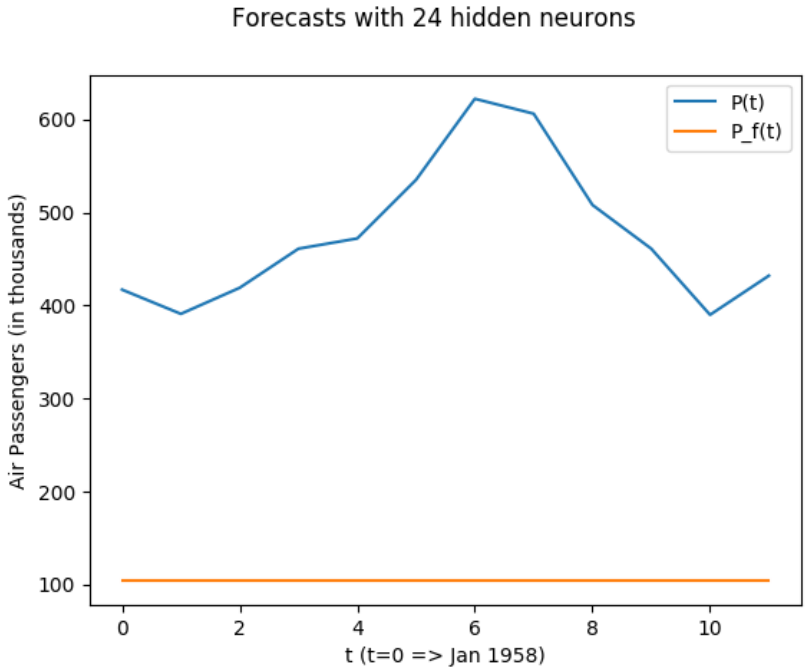
Recall that ARIMA is a well-established technique of forecasting time series (Section 2). Thus ARIMA could serve as a benchmark forthe `keras` LSTM

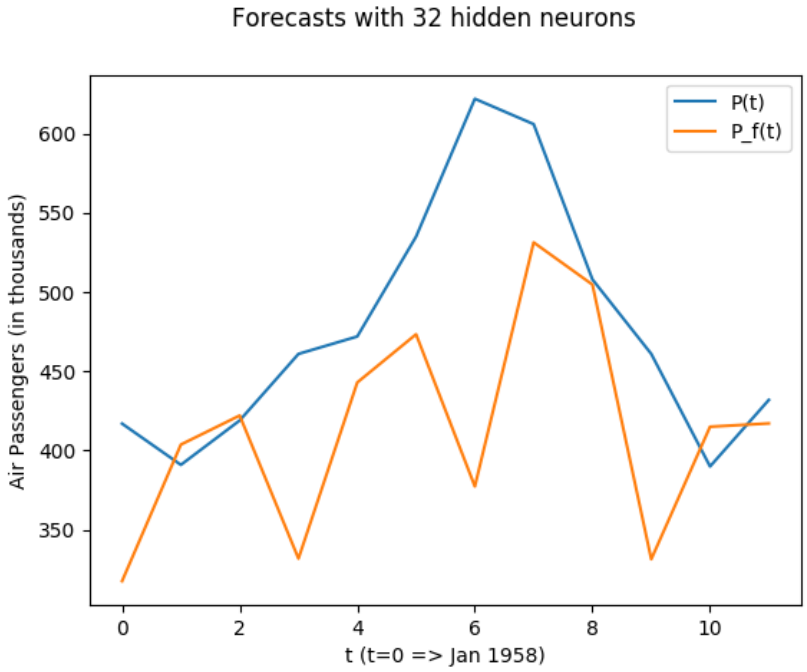
model With ARIMA, I obtained a MASE value of 0.82. Now, with `keras` LSTM model I obtained a range of MASE values, the best of which was 0.37. So it seems that `keras` LSTM can perform a little better than ARIMA.

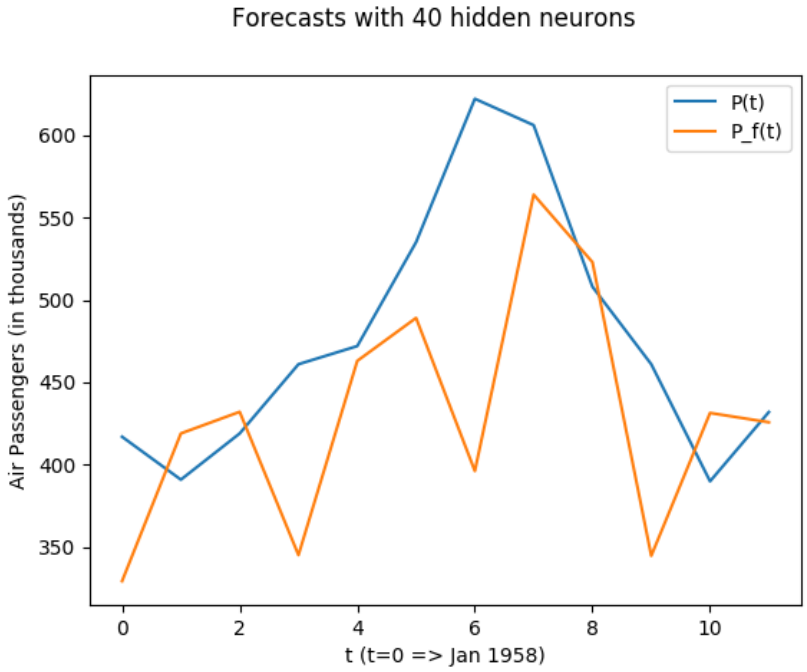
7.2 Graphs of Forecasts vs Exact Values

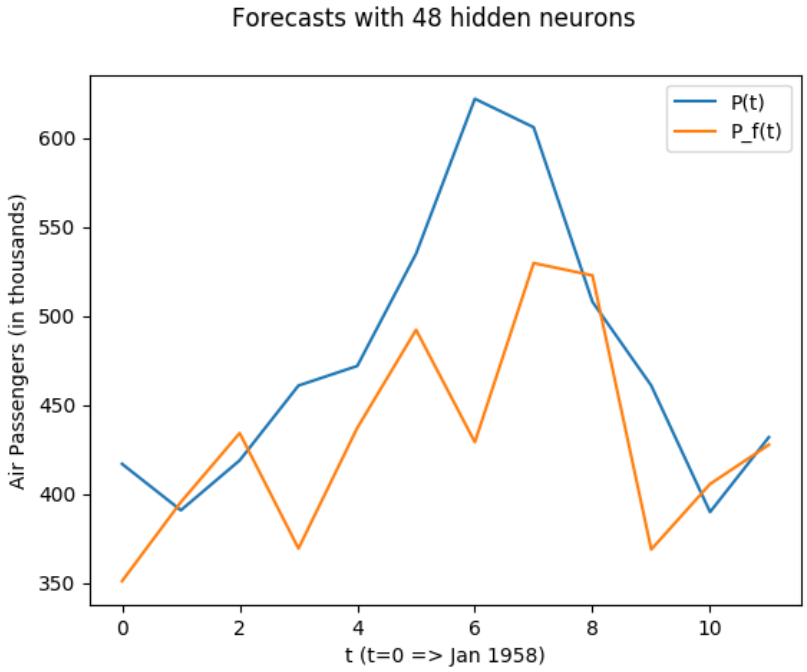
The graphs below compare $P_f(t)$ with $P(t)$.

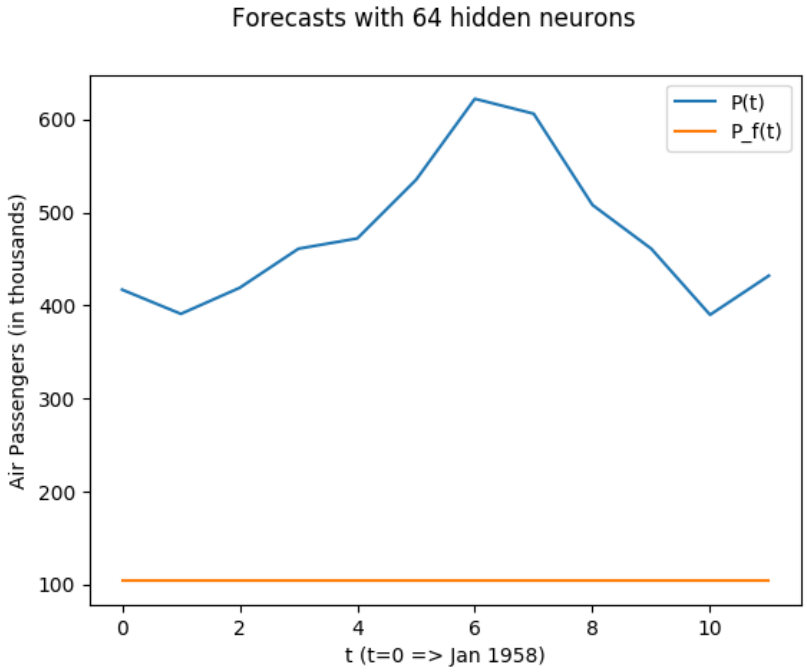


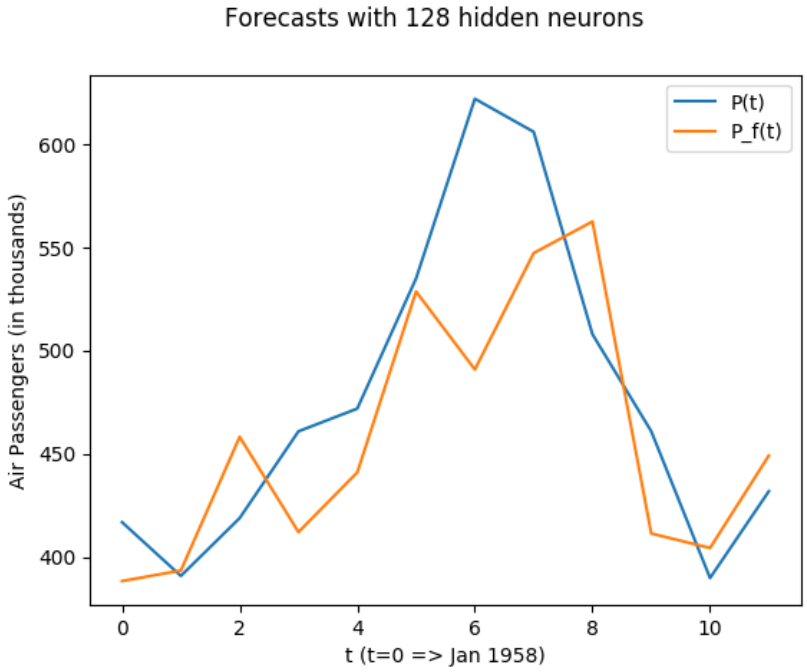

Forecasts with 24 hidden neurons
P(t)
P_f(t)
Air Passengers (in thousands)
600
500
400
300
200
100
0
2
4
6
8
10
t (t=0 => Jan 1958)

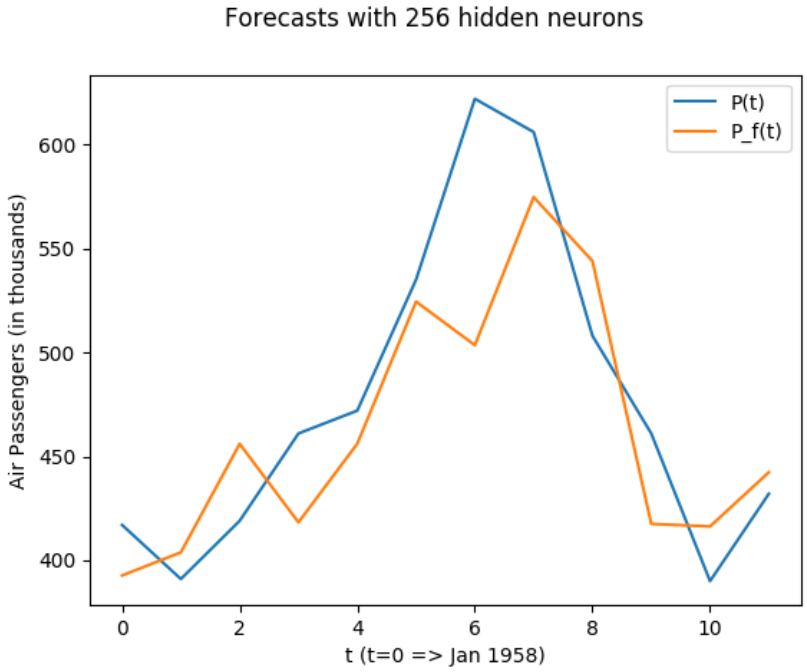

Forecasts with 32 hidden neurons
P(t)
P_f(t)
Air Passengers (in thousands)
600
550
500
450
400
350
0
2
4
6
8
10
t (t=0 => Jan 1958)

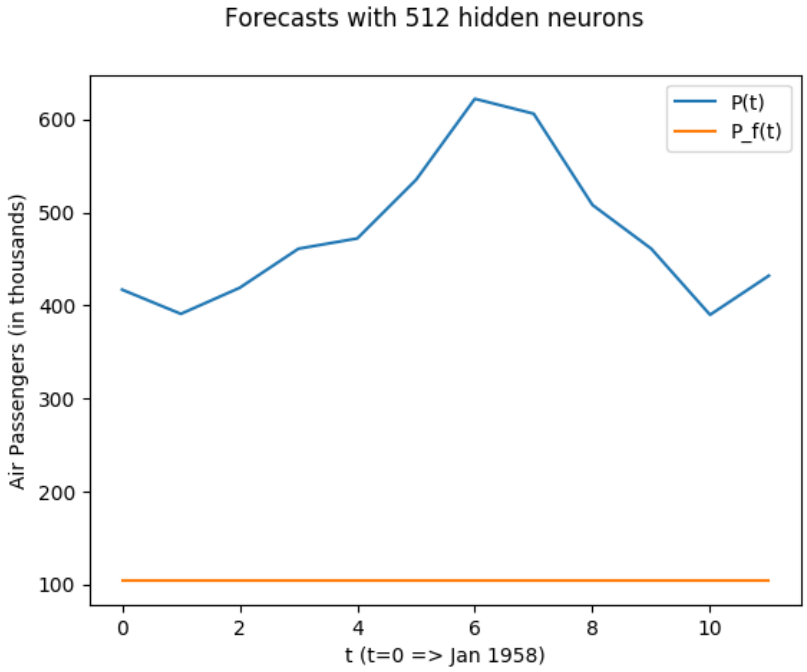

Forecasts with 40 hidden neurons
P(t)
P_f(t)
Air Passengers (in thousands)
600
550
500
450
400
350
0
2
4
6
8
10
t (t=0 => Jan 1958)


Forecasts with 48 hidden neurons
P(t)
P_f(t)
Air Passengers (in thousands)
600
550
500
450
400
350
0
2
4
6
8
10
t (t=0 => Jan 1958)


Forecasts with 64 hidden neurons
P(t)
P_f(t)
Air Passengers (in thousands)
600
500
400
300
200
100
0
2
4
6
8
10
t (t=0 => Jan 1958)

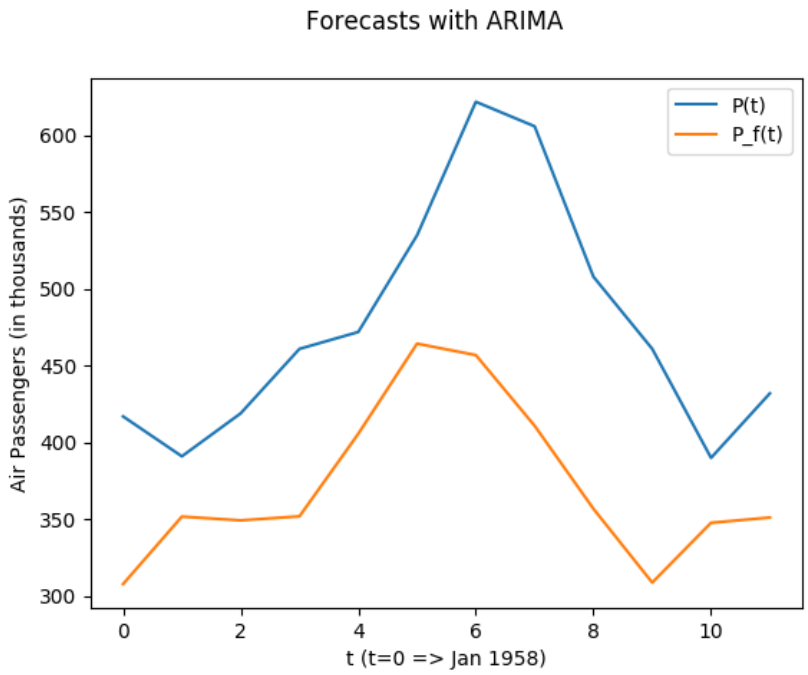

Forecasts with 128 hidden neurons
P(t)
P_f(t)
Air Passengers (in thousands)
600
550
500
450
400
0
2
4
6
8
10
t (t=0 => Jan 1958)





ARIMA as benchmark

As mentioned in Section 7.1, I am using ARIMA as a benchmark. Therefore let us look at the graph of an ARIMA forecast:



7.3 Robustness

What is robustness? It means that the results do not change much, if some of the input data are perturbed by a small amount—this is one criterion with which to judge robustness. Likewise, the results do not change much, if some of the adjustable parameters are perturbed by a small amount—this is a second criterion with which to judge robustness.

I am going to use the second criterion of robustness; the adjustable parameter is going to be the number of hidden neurons. The argument I will be making is that the results don't change much if the number of hidden neurons changes a bit.

In Section 7.1 the value of MASE was 0.37 when the number of hidden neurons was 256. If the number of hidden neurons was changed to 128, the value of MASE changed a little, to 0.43. If the number of hidden nuerons was changed further to 64, the value of MASE changed a little more, to 0.50. That is to say, the value of MASE changes continuously as a function of the number of hidden

neurons. By the second criterion of robustness, the solution presented here is robust.

7.4 A Further Comment on the Number of Hidden Neurons

Recall that the value of MASE is 0.37 when the number of hidden neurons is 256 (Section 7.1). Let us reflect on the number 256 from another angle.

Brownlee [3] also used a `keras` LSTM model for the airline passenger dataset. But he used merely 4 hidden neurons! Why, then, did I need 256? Why did I get poor results when I used 4 hidden neurons? I suggest the answer is as follows.

Recall that the code reads in the dataset into a `pandas` dataframe denoted as D_{orig}. Section 6.1.) D_{orig} contains a linear trend and a seasonal trend. Brownlee [3] removed the linear and seasonal trends. This operation created a new `pandas` dataframe denoted as $D_{detrend}$. Brownlee applied the `keras` LSTM model to $D_{detrend}$. Therefore his `keras` LSTM model did not have to model the linear or seasonal trends. In contrast, I applied the `keras` LSTM model to D_{orig}. Therefore my `keras` LSTM model did have to model the linear or seasonal trends.

Modeling a linear trend is not hard—it should not need a lot of hidden neurons. But modeling a seasonal trend does need a lot of hidden neurons. Hynk [5] used 300 hidden neurons in order to model a sine function. (A sine function is a simple example of a seasonal trend.) Aungiers [1] used two layers of hidden neurons in order to model a sine function, one layer with 50 hidden neurons and the other with 100 hidden neurons. On the basis of [5] [1], I suggest that 150–300 hidden neurons are required to model a seasonal trend. If this suggestion is accepted, then my figure of 256 hidden neurons is reasonable.

7.5 Reasonability or Trustworthiness of the Solution

The computed forecasts $P_f(t)$ pass some necessary conditions.

First the values of $|P_f(t)|$ are the right order of magnitude. Had they been much, much greater than $|P(t)|$, that would have been a red flag; Likewise had they been much, much lower than $|P(t)|$, that would have been a red flag, as well. In reality, the values of $|P_f(t)|$ are the right order of magnitude, so those red flags do not arise.

Second, the graphs of $P_f(t)$ continue the linear and seasonal trends that were

observed in the training set. Had $P_f(t)$ failed to continue the linear and seasonal trends, that would have been a red flag. In reality, this red flag does not arise.

Third, the value of MASE varies continuously as a function of the number of hidden neurons, indicating that the solution is robust. Had the solution not been robust, that would have been a red flag. In reality, this red flag does not arise.

Fourth, the number of hidden neurons in my code is comparable with what has been published in the literature (Section 7.4).

Fifth, the solution presented here is competitive with respect to ARIMA. The solution does not out-class ARIMA, though. That is reasonable because ARIMA has been used for some 40 years on many classes of time series, including this particular dataset.

A Missing Element

The solution presented here does not have cross-validation. It is an area for improvement (Section 9).

8 Difficulties I Encountered

The difficulties I encountered can be put under two heads.

First, I got error messages that said that this or that library was not available, or was available in the wrong version. It took a lot of hours to upgrade my laptop with appropriate versions of `keras, tensorflow` and `cudasolver`.

Second, I found it hard to relate the `keras` LSTM documentation to the dataset. More precisely, it took me a long time to figure the topic of reshaping `train_x` and `test_x` into three- dimensional arrays (Section 6.4).

9 Areas for Improvement

My solution does not have cross validation. Cross validation for a time series exists, but it is not quite the same as cross-validation for unordered datasets. Adding cross-validation to the solution would improve confidence in the robustness of the solution.

The theory of ARIMA is rich, so much so that ARIMA is able to add confidence intervals or error bars to its forecasts. Perhaps that theory could be imported

into the `keras` LSTM model, too.

10 Conclusion

Let us summarize what has been done in this project.

- The data set is a time series.
- I have examined whether the data set has missing data or nans—the answer was in the negative.
- I have done preliminary data exploration. The preliminary data exploration revealed that the dataset does no have outliers, and its distribution is far from the normal probability distribution. Further the dataset has a linear trend and a seasonal trend. The seasonal trend behaves like a periodic function of time with a time period of 12 months.
- I have formulated the problem statement as a forecasting problem, *i.e.,* to generate forecasts with a 36-month forecast horizon.
- I have shown how the problem statement can be modeled by means of a recurrent neural network.
- To implement a recurrent neural network in software, I used the *keras* LSTM model.
- Using the LSTM model I was able to generate forecasts with a 36- month forecast horizon.
- How accurate were these forecasts? To quantify the accuracy, I wrote code for an error metric named MASE which is recommended in the literature but is not available in a ready-made manner in `numpy, sklearn` or `pandas.`
- I was able to show that my implementation delivered forecasts that were comparable to the ones published in the literature using other methods such as ARIMA.
- I was able to demonstrate that the results were robust.
- I had a moment of exhiliration when I was able to understand why I used far more hidden neurons than Brownlee [3]. My hidden neurons had to model the seasonal trend, while Brownlee's did not. Brownlee detrended the seasonal trend, *i.e.,* Brownlee used an analytical model (not hidden neurons) in order to model the seasonal trend. There is a trade-off: One can run a detrend operation and use a small number of hidden neurons, or one can omit a detrend operation and use a large number of hidden neurons.

- I was able to point out areas for improvement, *i.e.,* I was able to generate problem statements for future work.

Appendices

A Advantages of the Dataset

The dataset is recognized for these properties:

- The dataset has an increasing linear trend
- The dataset has a seasonal trend
- The dataset has been modeled adequately by SARIMA. SARIMA a variant of ARIMA that is suitable for seasonal data. SARIMA can be used as a benchmark or a baseline against which we can compare an RNN's performance on the same dataset.
- The dataset is short, so the runtime is short. Therefore the student can fix software bugs quickly.

B More on MASE

B.1 How to Compute MASE

MASE is short for Mean Absolute Scaled Error [4]. It is computed thus:

$$\begin{aligned} \text{numerator} &= \text{mean}\{|P_f(s-h,s) - P(s)|, s, s-h \in \text{testing set}\} \\ \text{denominator} &= \text{mean}\{|P(t+h) - P(t)|, t, t-h \in \text{training set}\} \\ \text{MASE} &= \frac{\text{numerator}}{\text{denominator}} \end{aligned}$$

To understand this formula, let us consider a toy example. Let the training set comprise 12 data points:

$$\begin{aligned}
P_f(\text{Jan 1949}) &= 112,000 \\
P_f(\text{Feb 1949}) &= 118,000 \\
P_f(\text{Mar 1949}) &= 132,000 \\
P_f(\text{Apr 1949}) &= 129,000 \\
P_f(\text{May 1949}) &= 121,000 \\
P_f(\text{Jun 1949}) &= 135,000 \\
P_f(\text{Jul 1949}) &= 148,000 \\
P_f(\text{Aug 1949}) &= 148,000 \\
P_f(\text{Sep 1949}) &= 136,000 \\
P_f(\text{Oct 1949}) &= 119,000 \\
P_f(\text{Nov 1949}) &= 104,000 \\
P_f(\text{Dec 1949}) &= 118,000
\end{aligned}$$

Let the forecast horizon be h = 2 months.

The denominator of MASE is the mean of the following 10 quantities:

$$\begin{aligned}
|P_f(\text{Mar 1949}) - P_f(\text{Jan 1949})| &= |132,000 - 112,000| = 20,000 \\
|P_f(\text{Apr 1949}) - P_f(\text{Feb 1949})| &= |129,000 - 118,000| = 11,000 \\
|P_f(\text{May 1949}) - P_f(\text{Mar 1949})| &= |121,000 - 132,000| = 11,000 \\
|P_f(\text{Jun 1949}) - P_f(\text{Apr 1949})| &= |135,000 - 129,000| = 06,000 \\
|P_f(\text{Jul 1949}) - P_f(\text{May 1949})| &= |148,000 - 121,000| = 27,000 \\
|P_f(\text{Aug 1949}) - P_f(\text{Jun 1949})| &= |148,000 - 135,000| = 13,000 \\
|P_f(\text{Sep 1949}) - P_f(\text{Jul 1949})| &= |136,000 - 148,000| = 08,000 \\
|P_f(\text{Oct 1949}) - P_f(\text{Aug 1949})| &= |119,000 - 148,000| = 29,000 \\
|P_f(\text{Nov 1949}) - P_f(\text{Sep 1949})| &= |104,000 - 136,000| = 32,000 \\
|P_f(\text{Dec 1949}) - P_f(\text{Oct 1949})| &= |118,000 - 119,000| = 01,000
\end{aligned}$$

Therefore

$$\text{denominator} = \frac{20,000 + 11,000 + 11,000 + \ldots + 01,000}{10} = 15,800$$

Further let the testing set comprise these four months: Jan 1950, Feb 1950, Mar 1950, Apr 1950. What this means is that in the notation $P_f(s-h, s)$, we will set s = Jan 1950, Feb 1950, Mar 1950, Apr 1950. And we will set $s - h = s -$ two months = Nov 1949, Dec 1949, Jan 1950, Feb 1950. Let our forecasting algorithm compute the following forecasts:

Month s	Forecast $P_f(s-h,s)$	Exact Value $P(s)$
Jan 1950	120,000	115,000
Feb 1950	125,000	126,000
Mar 1950	133,000	141,000
Apr 1950	140,000	135,000

The numerator of MASE is the mean of these four numbers:

$$|120,000 - 115,000| = 5,000$$
$$|125,000 - 126,000| = 1,000$$
$$|133,000 - 141,000| = 8,000$$
$$|140,000 - 135,000| = 5,000$$

Therefore

$$\text{numerator} = \frac{5,000 + 1,000 + 8,000 + 5,000}{4} = 4,375$$

Therefore

$$\text{MASE} = \frac{\text{numerator}}{\text{denominator}} = \frac{4,375}{15,800} = 0.28$$

B.2 Scale Independence or Unit Independence

Let us recall that the unit of $P(t)$ is passengers. Suppose that the unit were changed to thousands of passengers. The change of unit would have an impact on the value of $P(t)$. For example, suppose that $P(\text{Dec } 1960) = 432,000$ with the unit set to passengers. When the unit is changed to thousands of passengers, $P(\text{Dec } 1960) = 432$.

The forecast $P_f(t, t+h)$ is also affected by the change of unit, in the same way as $P(t)$.

However, the value of MASE is the same, no matter what the unit is. For example, let MASE = 0.36 with the unit set to passengers. When the unit is changed to thousands of passeners, MASE will continue to be 0.36. That is to say, MASE has the property of scale independence or unit independence.

B.3 Does Not Need Time Series to Be Bounded Away from Zero

We say that $P(t)$ is bounded away from zero if $P(t) \neq 0$ for all values of t; in fact, there should be a positive number, say, 0.0001 such that $|P(t)| \geq 0.0001$ for all values of t.

Likewise, we say that $P_f(t,s)$ is bounded away from zero if $P_f(t,s) \neq 0$ for all values of t, s; in fact, there should be some value, say, 0.007, such that $|P_f(t,s)| \geq 0.007$ for all values of t, s.

Some error metrics work well only if $P(t)$ and $P_f(t,s)$ are bounded away from zero. No so MASE; MASE works well even if $P(t)$ and $P_f(t,s)$ attain the value 0, or come infinitesimally close to 0.

It must be admitted, though, that $P(t)$ is bounded away from 0 in the present dataset (Series G of [2]), so this advantage is academic in the present dataset.

B.4 A Limitation of MASE

Suppose that

$$P(\text{Jan 1949}) = P(\text{Feb 1949}) = P(\text{Mar 1949}) = \ldots P(\text{Dec 1960}) = 112,000$$

Then the denominator of MASE's formula would be 0, and MASE would be undefined. This problem happens in one case only, namely, when the time series is a constant function of time—more precisely, when the time series is constant on the training set.

References

[1] J. Aungiers, *LSTM Neural Networks for Time Series Prediction—IoT Data Science Conference,* accessed at 3:28 pm PDT on July 7, 2017

[2] G.E.P. Box & G.M. Jenkins, *Time Series Analysis Forecasting and Control,* Holden Day 1976

[3] J. Brownlee, *Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras,* http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/ accessed at 6:53 pm PDT on June 2, 2017

[4] R.J. Hyndman & A.B. Koehler, *Another Look at Measures of Forecast Accuracy,* International Journal of Forecasting, 22(4), p679, 2006 https://robjhyndman.com/papers/mase.pdf accessed at 1:54 pm PDT on June 5, 2017

[5] D. Hynk, *Predicting sequences of vectors (regression) in Keras using RNN—LSTM,* http://danielhnyk.cz/predicting-sequences-vectors-keras-using-rnn-lstm/ accessed at 3:18 pm PDT on July 7, 2017