# Sky Cast: A Comparison of Modern Techniques for Forecasting Time Series

Franklin Bradfield

January 2018

## 1 Definition

### 1.1 Project Overview

The human race has a long history of trying to predict the future, occasionally successfully, but more often than not to little avail. Of all the diverse fields that machine learning has been introduced to the past several decades, time series forecasting stands out as one of the most prominent and impactful for practitioners and stakeholders alike. Time series forecasting is concerned with predicting the future behavior of a random variable based on its past behavior as well as exogenous factors where available. It assumes that there are one or more patterns in a given random variable that emerge over time that can be quantified and modeled. Once a model's parameters are fitted to past data, it can in turn be used for predicting direction out to various points in the future. Examples of common patterns include trends, seasonality, and Gaussian noise. Businesses don't have to look far to find clear use cases where machine learning can prove itself as a powerful tool that they can apply to solve problems involving time series. Demand and cost forecasting are immediately obvious, however in principal machine learning for prediction could be useful in any area where a business keeps historical data. Time series data is all around us, therefore being able to predict it accurately remains a goal to which many researchers have applied their efforts. A few of the fields in which time series forecasting has proven invaluable include:

- Finance [1] [2] [5]
- Energy [3]
- Healthcare [6]
- Meteorology [7]

Prediction is not a novel problem and certainly not one for which machine learning is the only useful tool. Time series forecasting is a part of the broader field of predictive analytics, which includes machine learning techniques, but also utilizes a variety of traditional statistical approaches as well. Autoregressive integrated moving average (ARIMA) is a regression technique that is frequently applied in finance to predict the future value of financial assets [2]. Deep learning on the other hand has also been used to successfully forecast time series data. Traditional multilayer-perceptrons (MLPs) have been employed with varying degrees of success in this domain [3] [4] [5] [7], but more often than not the tools of choice for time series data are different variations of recurrent neural networks (RNNs), namely the long short-term memory (LSTM) network [6], which excels in capturing both long and short-term dependencies between time steps.

Recurrent neural networks are famous for their use in natural language processing, which can be viewed as another manifestation of the class of sequence problems, of which time series forecasting is a member. Deep learning approaches have demonstrated superior results on time series problems as compared with statistical methods [3] [5], however this pattern doesn't always hold, for example in more simple cases where short-term dependencies are all that is needed to make predictions [4]. This report further discusses how statistical methods and deep learning differ fundamentally in the way they model and forecast sequence data, as understanding this will highlight the types of problems for which each approach is most suitable.

The primary motivation for this project is to add to the body of research that compares traditional statistical methods with deep learning techniques to time series forecasting. The commercial aviation industry is used as the testing ground for comparing ARIMA and RNNs on time series data. Between statistical methods and neural networks, the question of which approach is "better" for modeling and forecasting time series has not yet been answered conclusively. It may be that the proper choice of model is wholly dependent on the specific problem one is trying to solve, and that one should consider the various benefits and trade-offs of each approach before settling on a choice. To this end, limitations and drawbacks of each model are also discussed and various aspects of time series problems that validate or invalidate the use of certain approaches are examined.

No public research was found in the literature that involves neural networks being applied for forecasting in this particular field, although major airlines themselves certainly take time series forecasting seriously within their organizations. With that, there are a variety of business applications that such research can provide as well. In an industry where excess capacity equals wasted expenditure and where insufficient capacity equals lost revenue, it is imperative for airlines to be able to forecast demand with the greatest degree of accuracy as possible. Having robust prediction models for quantities such as next year's passenger count could ensure that sufficient resources are allocated in advance to handle incoming demand. This is but one example of the practical applications of this project.

## 1.2 Problem Statement

The question that this project seeks to answer can be detailed as such:

- Can future data in the commercial aviation industry be accurately forecasted based solely on past data from the same series using techniques from time series regression and/or deep learning?.

Specifically, the task is to make yearly forecasts on four industry dimensions of interest by predicting time series from April 2016 to March 2017 using only data from October 2002 to March 2016 for training and validation. The three time series datasets selected for testing include American Airlines at Dallas/Fort-Worth (AA-DFW), Delta Airlines at Atlanta (DL-ATL), and United Airlines at Chicago (UA-ORD). For each individual series, the task

| *Type* | *Definition* |
|---|---|
| Passengers | Number of passengers that flew in the month |
| Flights | Number of flights that were conducted in the month |
| ASM | Number of available seats * Number of miles flown |
| RPM | Number of revenue paying passengers * Number of miles flown |

Table 1: Definition of the industry's time series dimensions

can thus be viewed as a univariate time series problem, where models take 162 time series values as input and forecast 12 more that reflect learned patterns in the series, such as trend, seasonality, residual noise, etc. In a real-world scenario, training models with multiple time series inputs, known as multivariate time series, might well improve forecast accuracy but that is not covered in the scope of this project. Reasonable additional features that could be added are mentioned in the Section 5.3 (improvement).

There are four dimensions of industry data that the datasets contain for each month: the number of passengers, number of flights, total available seat-miles (ASM), and total revenue passenger-miles (RPM). This leads to four separate models and four prediction scores for any airline-airport combination. Although data on each type is reported on a discrete scale, in practice making these forecasts is treated as a regression problem because the values typically reach the tens or hundreds of thousands. As such, fractional values that arise in final forecasts will be rounded to the nearest whole number. More specific definitions of the four industry dimensions are included in the Table 1.

The data used in this project is not static, with new additions being added by the source once a month. This creates an ongoing opportunity for the performance of models to be evaluated so that improvements can be tested and applied as they are devised. Although this project specifically focuses on commercial aviation data, it is further intended to serve as a proof-of-concept for the use of RNNs applied to time series data in general, as their success in this domain is still emerging in the literature. If their results are successful, then this will provide further support for choosing RNNs on a variety of time series problems.

## 1.3 Metrics

In the project proposal, a combination of Root Mean-Squared Error (RMSE) and R-Squared ($R^2$) was put forth to the measure the models' performance, however both of these measurements proved inadequate for valid comparison between ARIMA and RNNs. The original justification for supplementing RMSE with $R^2$ was that the four dimensions of the data (Passengers, Flights, ASM, RPM), are measured on different scales, thereby preventing an apples-to-apples comparison of their forecasts. However, as will be explained more in the Section 3.1, the data was standardized with zero mean and unit variance prior to training, which sharply exaggerated the $R^2$, thus rendering it an uninterpretable metric. As a result, a new metric was sought altogether that could overcome the deficiencies in both of these metrics - the Mean Absolute Percentage Error (MAPE):

$$\text{M} = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

Where $A_t$ is the actual value and $F_t$ is the forecast value. MAPE has the advantage of being easy to interpret - its simply the average percentage that a point-by-point forecast deviates from ground truth, while also being suitable for comparing performance between models that optimize against different objective criteria. In this case, ARIMA minimizes the Akaike Information Criterion (AIC) through maximum likelihood estimation (MLE) of the best model parameters, while RNNs can minimize a variety of objective functions with gradient-based optimization, therefore MAPE helps directly compare results of their forecasts.

## 2 Analysis

### 2.1 Data Exploration

There are 12 time series that are of interest for forecasting in this project, covering the four dimensions and three airline-airport combinations. As mentioned, the four industry dimensions that are forecasted are monthly passengers, flights, and available seat-miles revenue passenger-miles. Refer to Section 1.2 for a definition of these terms. The three airline-airport pairs of interest are American-Dallas/Fort-Worth, Delta-Atlanta, and United-Chicago. Each series includes 174 monthly values.

An import aspect of time series data that must be taken into account when forecasting is the degree of stationarity that is present. This is especially relevant when forecasting using statistical models such as ARIMA, which perform better when applied to stationarity data. Generally speaking, stationarity refers to the degree to which a time series' mean and variance remain constant in time. Trends such as growth or decline, and seasonality are two common features of time series data that cause non-stationarity, i.e. shift in mean and variance across time. A quick glance at the time series' raw values and moving averages demonstrates both trend and seasonality at play, therefore before proceeding with forecasting, it is necessary to quantify the degree of non-stationarity.

Developed in 1979 by David Dickey and Wayne Fuller, the Dickey-Fuller Test provides a means to measure stationarity in a given time series. It is not the goal to explain the technicalities of this hypothesis test - refer to [8] for its precise inner workings. For the project's purposes, the lower the p-value that the test outputs, the higher degree of stationarity in the time series. This test was performed for the raw passengers time series for the three airport-airline pairs in the data_analysis Jupyter Notebook, and the results are summarized in Table 2. With p-values in excess of 0.50 at the 5% significance level, non-stationarity is significantly present in both the Delta-Atlanta and United-Chicago passenger time series. The test statistic for the American-Dallas series is low enough to be significant even at the 1% level, indicating strong stationarity. Time series visualizations in Section 2.2 support these results as well.

The differencing parameter of ARIMA provides a means of rendering a time series stationary through calculating differences between previous values. This will further be explained in Section 2.3, but for now it is sufficient to state that differencing can aid in reducing trend but not seasonality. That on the other hand must be extracted manually, or it can be handled more effectively with more advanced regression models, such as seasonal ARIMA. The utils.py file contains Python functions that attempt to manually remove and add back

| | American-Dallas | Delta-Atlanta | United-Chicago |
|---|---|---|---|
| Critical Value 1% | -3.472 | -3.472 | -3.472 |
| Critical Value 5% | -2.880 | -2.880 | -2.880 |
| Critical Value 10% | -2.577 | -2.577 | -2.577 |
| Test Statistic | -3.760 | -0.2425 | -1.350 |
| P-Value | **0.003** | 0.9333 | 0.6059 |

Table 2: Dickey-Fuller test results on raw passenger series

| | American-Dallas | Delta-Atlanta | United-Chicago |
|---|---|---|---|
| Test Statistic | -2.2250 | -2.1868 | -1.1874 |
| P-Value | 0.1974 | 0.2111 | 0.6790 |

Table 3: Dickey-Fuller test results on deseasoned passenger series

seasonality. After applying the season removal function to the three passenger time series, the Dickey-Fully test was conducted again to determine if it could increase stationarity. The subsequent test statistic and p-values that were generated are displayed in Table 3.

The results here are mixed, with stationarity increasing in Delta-Atlanta (through lower p-value), but decreasing in American-Dallas (through higher p-value). United-Chicago is mostly unchanged in this regard. Overall, removing seasonality does appear effective in increasing stationarity enough on some time series to justify its use in the data preprocessing step for ARIMA. The difference operator in ARIMA should help offset the slight decrease in stationarity that this step incurs against other series. There's also an empirical basis for removing seasonality in that it is necessary to allow ARIMA to converge during training on these time series. This is returned to when discussing data preprocessing in Section 3.1.

### 2.2 Exploratory Visualization

In the interest of space and brevity, only trend plots for monthly passengers are examined here in Figure 1 - 3. It's worth noting that the series for monthly flights, ASM, and RPM are similar in their shape and structure - as it is fundamental passenger demand that drives these other three dimensions. Autocorrelation plots are produced and examined in the following section, as these are integral in choosing the right model parameters for the algorithm that ARIMA implements.

The trend plots of the three raw time series demonstrate several notable characteristics worth discussing. Each series contains clear seasonality, as evidenced by the rolling peaks and valleys that continue from end to end. Shown in the 12-month rolling means (red line), American-Dallas/Fort-Worth begins with a growth trend and holds sideways for the rest of its duration. After experiencing a slump during the 2007-2008 financial crisis, Delta-Atlanta carries a growth trend for much of its length. Lastly, United-Chicago consists of a downtrend which reverses to an uptrend near its end. Sudden shifts in trend such as this could prove difficult to forecast. An additional feature that is observable throughout each time series is heteroskedasticity, i.e. non-constant variance. This is demonstrated in the varying steepness of the seasons' peaks and valleys. For example in United-Chicago, the differences are drastic at the beginning, before softening, and then finally increasing again

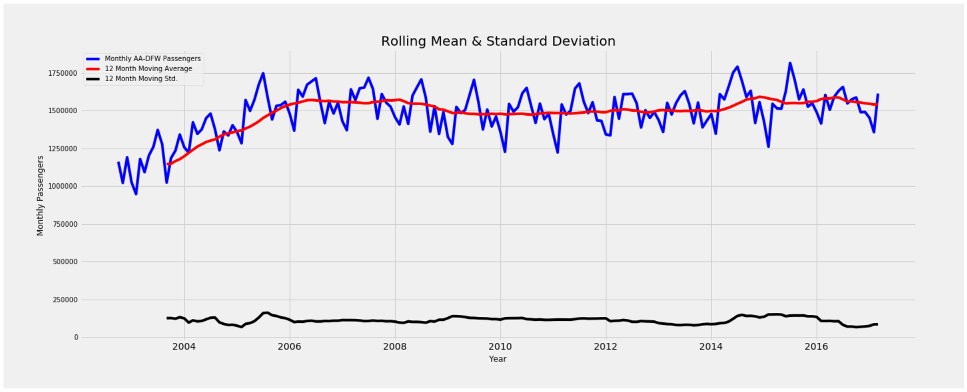

Figure 1: Trend Plot of AA-DFW

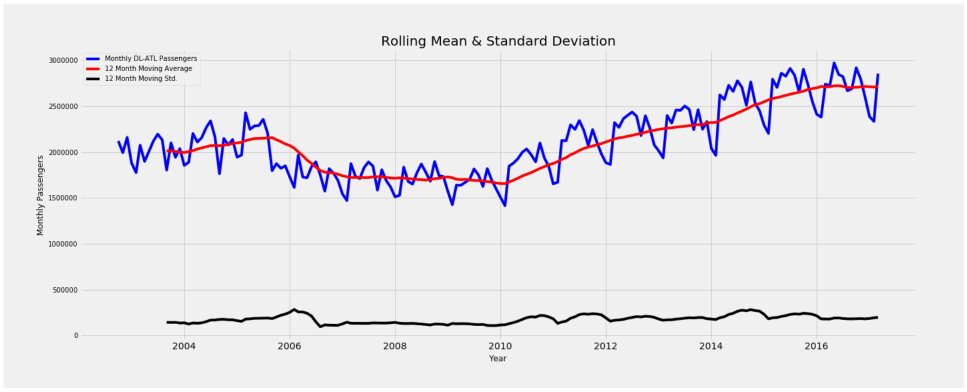

Figure 2: Trend Plot of DL-ATL

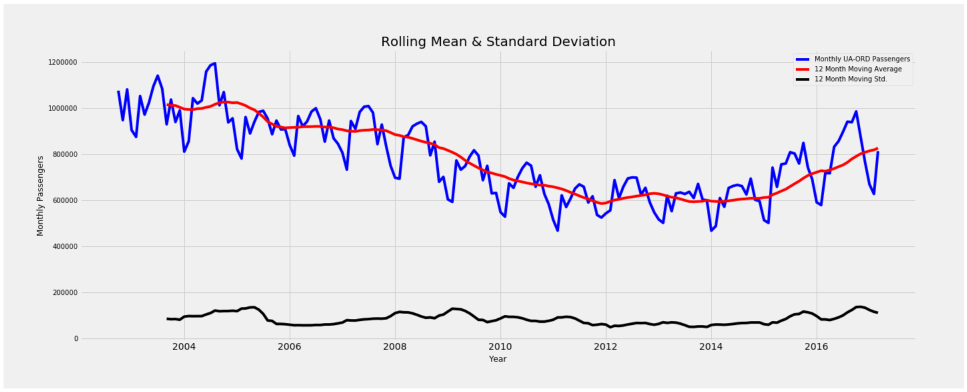

Figure 3: Trend Plot of UA-ORD

in the final year or so. This feature is quantitatively captured in the plots' 12-month rolling standard deviations (black line), which is smooth in some areas and bumpier in others. Heteroskedasticity could cause difficulties for the models as, unlike trend, none of them have settings that can address it directly. Nevertheless, far from being totally stochastic, each series shows signs of repeating patterns that should be able to captured and forecasted with advanced enough models. Unlike other time series in the dataset at large, none of the series contain extreme spikes or drop-offs that are impossible to model and predict without access to external information.

## 2.3 Algorithms and Techniques

ARIMA is a statistical regression model that involves regressing weight parameters on a time series' lagged values. It extends less advanced regression methods such as ARMA by including a differencing step that helps to smooth out non-stationarity and increase forecasting accuracy. The differencing mechanism is explained further - with an illustrative example - in Section 3.1. This model takes three parameters - $\{\boldsymbol{p}, \boldsymbol{d}, \boldsymbol{q}\}$ - and is fitted to time series data and then used to make predictions. Briefly, $\boldsymbol{p}$ determines the number of time series lags (historical values) to regress against, $\boldsymbol{d}$ determines how many times to difference the time series, and $\boldsymbol{q}$ determines the number of (normally distributed) error terms to regress against.

One drawback of ARIMA is that the user must estimate the optimal values for $\{\boldsymbol{p}, \boldsymbol{d}, \boldsymbol{q}\}$ prior to fitting. This often requires a strong grasp of the underlying statistics if one wants to produce good results. He or she could also perform exhaustive or random search to find optimal parameters, but there are still various issues such as trend, seasonality, and noise that must be taken into account. For these reasons, ARIMA is not a kind of "plug-and-play" black-box model that a beginner can pick up and start working with quickly. It also becomes more and more expensive to train ARIMA models as $\boldsymbol{p}$ is increased. Beyond $\boldsymbol{p}$ = 18, ARIMA becomes prohibitively expensive to train, but perhaps better hardware could mitigate this issue.

A quick method to determine an optimal lag parameter $\boldsymbol{p}$ to start from is by looking at time series' autocorrelation plots, such as those in Figure 4 - 6. These plots illustrate how much a time series correlates with its own lags. The value of $\boldsymbol{p}$ that should be chosen corresponds to the lag prior to a steep drop-off in autocorrelation. It can be seen in each plot that this point is 12 - where confidence is nearly 0.75 or greater - indicating that a given time series point can be roughly approximated as a function of its values from the previous 12 months. With this established, $\boldsymbol{p}$ of 12 is used as the starting point for ARIMA in this project.

A popular approach to forecasting time series with RNNs treats it as supervised regression problem, where the model trains on sliding windows of the series of interest and applies the parameters that it learns to unseen windows to make forecasts [12] [13]. As an example, take the series $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, where the goal is to predict one value into the future. The sliding-window approach with length = 5 would split the series into fixed-length, overlapping windows and represent the first four values in each as features and the last as label, e.g: $\{1, 2, 3, 4\} \rightarrow 5$, $\{2, 3, 4, 5\} \rightarrow 6$, $\{3, 4, 5, 6\} \rightarrow 7$, $\{4, 5, 6, 7\} \rightarrow 8$, $\{5, 6, 7, 8\} \rightarrow 9$, and $\{6, 7, 8, 9\} \rightarrow 10$. During testing, the window $\{7, 8, 9, 10\}$ could be fed into the model and it would (hopefully) predict *11*. As for forecasting multiple values, proponents of this approach simply add recursion to the process. Therefore if the

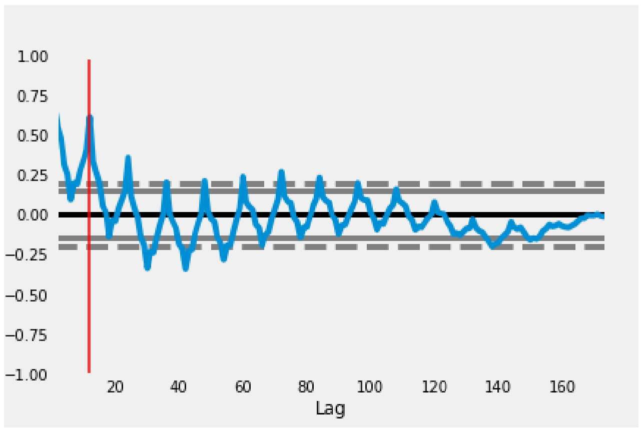

Figure 4: Autocorrelation Plot of American Passengers at DFW

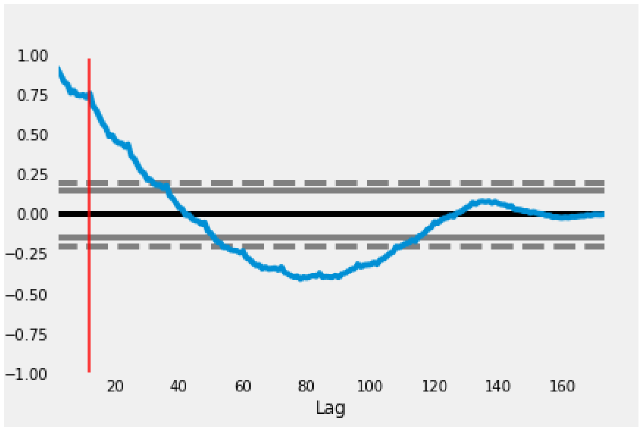

Figure 5: Autocorrelation Plot of Delta Flights at ATL

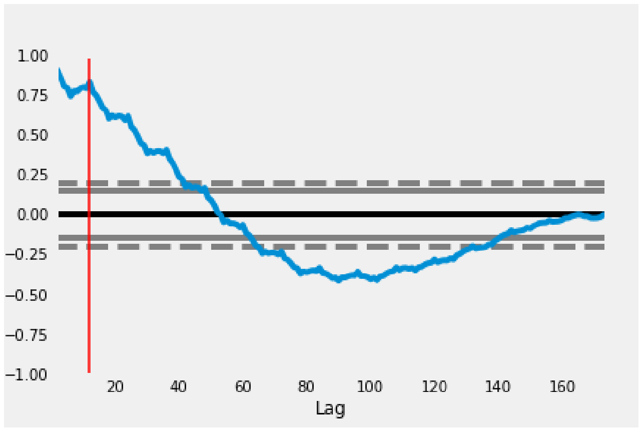

Figure 6: Autocorrelation Plot of United ASM at ORD

goal was to predict the next two values that follow {*7, 8, 9, 10*}, then the model would first predict *11* from {*7, 8, 9, 10*} and append this to a new testing window {*8, 9, 10, 11*}, and (once again hopefully) predict *12*. While this approach works moderately well on simple time series with plenty of sliding window data, it falls apart when almost any degree of complexity is introduced, especially when predicting multiple values is involved. After predicting two or three values into the future, the neural network quickly becomes unstable and starts making constant predictions that don't resemble the original series at all [12]. After experiencing the failure of this approach on the time series data in this project, it was clear that an enhanced approach had to be sought. For that, inspiration was taken from the winning solution to a recent Kaggle competition involving web time series [11].

In that competition, participants were required to build time series models for approximately 145,000 series from Wikipedia articles. Each series contained 550 training values, and the testing period lasted three months, therefore submissions required predicting 62 days into the future - a task loads more difficult than this project indeed. The winning solution to this problem in essence involved a multivariate Seq2Seq RNN, augmented with an attention interface - a highly modified and enhanced version of the model in Figure 7. Treating these kind of forecasting problems as sequence-to-sequence tasks, as opposed to recursive supervised tasks, makes a lot of sense, because indeed a training sequence serves as the input and a forecast sequence represents the output. This insight helped overcome the multiple output problem.

Figure 7 demonstrates the inner workings of a basic, single-layer Seq2Seq RNN. These models have proven invaluable in natural language processing tasks that involve incoming and outgoing sequences, such as machine translation, however they can work well for time series problems as well [11]. This model takes a sequence as input and is trained to output a sequence, potentially overcoming the multi-step prediction problem that plagued the previously discussed approach. In a nutshell, a Seq2Seq RNN is composed of two side-by-side RNNs, termed the encoder and decoder respectively. The two sides serve opposite purposes. As in a vanilla RNN, the encoder progressively composes a hidden state from successive inputs, however in this case, the final hidden state (encoder state) - a dense vector - isn't squashed in an activation function at the end for prediction. Instead the encoder network passes it to the decoder, which in turn "unrolls" it across a certain number of time steps. During training, the decoder can take only previous hidden states as input, or hidden states can be combined with the true value sequence so the network can more easily learn what it should have predicted at each step.

The particular version in Figure 7 uses Gated Recurrent Unit (GRU) cells at each time step, however basic RNN and LSTM cells could work (with possibly varying performance) in these positions as well. This project uses GRU cells in its implementation as well, primarily because they've been shown to attain similar performance as LSTMs, while involving less parameters and thus being more computationally efficient [9]. Here, the encoder network and decoder networks are 162 and 12 time steps long respectively, with each step corresponding to one month in the input time series. Additional hyperparameters such as number of hidden layers, number of hidden units, etc. are discussed in Section 3.3.

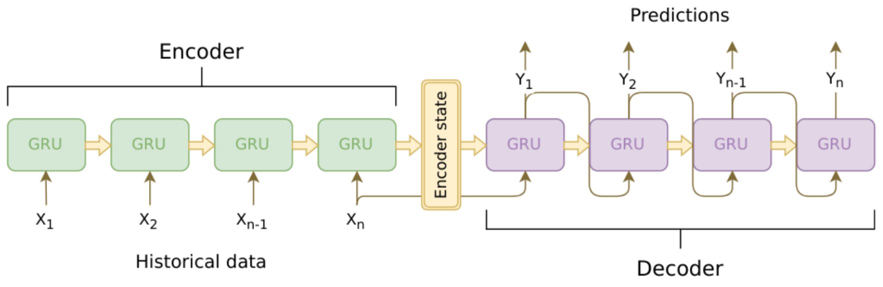

Figure 7: A basic Seq2Seq RNN (`https://github.com/Arturus/kaggle-web-traffic`)

| Dimension | American-Dallas | Delta-Atlanta | United-Chicago | Avg. |
|---|---|---|---|---|
| Passengers | 4.531 | 5.773 | 12.959 | 7.754 |
| Flights | 3.401 | 4.660 | 10.351 | 6.140 |
| ASM | 3.292 | 4.832 | 9.182 | 5.769 |
| RPM | 4.975 | 5.397 | 10.522 | 6.965 |

Table 4: MAPE for Median Models

## 2.4 Benchmark

The benchmark forecast model - ”the median model” - that was chosen involves projecting the median values in each raw time series from the final twelve training twelve months into the future. In other words, for each series the median from April 2015 to March 2016 was simply repeated at a constant rate from April 2016 to March 2017. This model was chosen for its simplicity, interpretability, and logicality. If a given time series contains a trend, then it makes sense that that trend would continue into the near future (next 12 months) - therefore projecting it’s twelve-month median is a simple way to forecast a time series while remaining conservative as to its trajectory. Table 4 shows the summary of results from the median models, where each cell contains the MAPE for a given series.

The median model produced surprisingly good results, supporting the intuition that existing trends persist into the future. Since MAPE is an error metric, the lower the scores the better. Across all dimensions, the most accurate forecasts were obtained for American-Dallas/Fort-Worth - likely due to its high degree of stationarity shown in Table Table 2. The best score among these was MAPE of 3.292 for available seat-miles - indicating an 3.292% average absolute deviation from the true values of a monthly forecast from April 2016 to March 2017. A diagram of this forecast is shown in Figure 8. Another interesting result is the stark differences in MAPE between United-Chicago and the other two series. As it turns out, this particular series is particularly difficult for all models to handle, for the reasons discussed in Section 2.2.

In Section 4.1, the MAPE results of forecasts from ARIMA and RNNs are reported and compared to these benchmark scores. One of the goals of this project is to outperform such simple models and provide support for employing state-of-the-art sequence models to

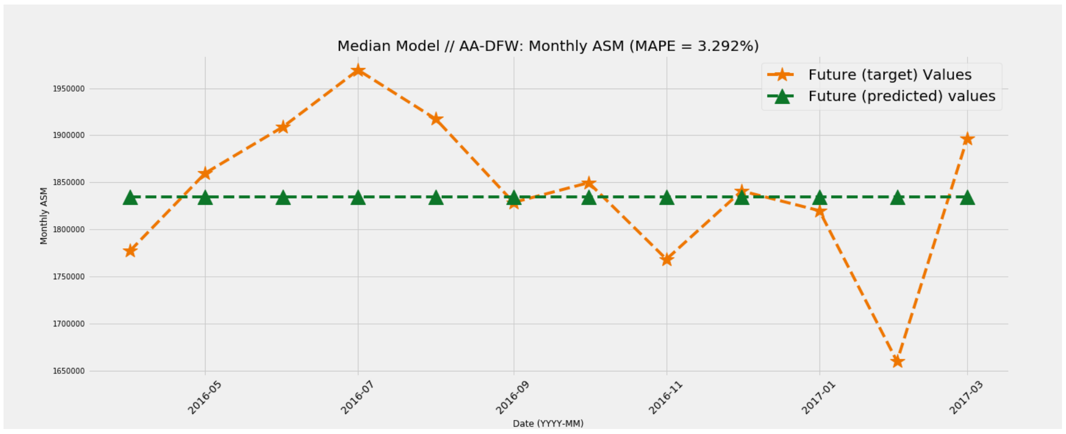

Figure 8: Median model forecast for ASM for American Airlines at DFW

generate strong results on time series problems such as this one.

## 3 Methodology

### 3.1 Data Preprocessing

Various data preprocessing procedures were undertaken prior to modeling and forecasting the time series with ARIMA and RNNs. The same train-test splits were performed for the median models, ARIMA and RNNs, whereby data from the first 162 months in each 174 month time series was set aside for training, leaving the final 12 months for testing - ”the ground truth”. For median models, no preprocessing was applied, because they simply involve projecting a statistical parameter and cannot apparently benefit from any such measures. The following preprocessing steps were performed only on training data - test splits were left in their raw forms. Each step was reverted in a ”post-processing” step that brought the forecasted values back to their original scale. In this way, forecasts could be directly compared to raw observed values with MAPE.

Prior to fitting each time series with ARIMA, the data was made as stationarity as possible so that ARIMA could do its best to only model the underlying residuals, and avoid getting hindered by trend and seasonality. To remove seasonality, a function was applied to each series that generates a new series where each value becomes the difference between its original value and that of 12 months prior. For example, if the monthly flights in April 2012 is 4,000 and April 2013 is 6,000, then the new value for April 2013 would be 2,000, as it is now the difference between its original value and that of the previous year. This reduces seasonality because it causes the series to effectively become a year-on-year trend. After removing seasonality, ARIMA was able to consistently perform 2 to 5 MAPE percentage points higher than it could otherwise for the various time series. Furthermore, ARIMA’s maximum likelihood estimation failed to converge on each flight series when seasonality was still present, resulting in MAPE in excess of 100% - rendering this step necessary for presentable results. Following fitting ARIMA, an inverse function was applied that returned seasonality to each series. As an example, if a residual of 2,000 passengers was forecasted for October 2014 and we observed 15,000 passengers in October 2013, then the "real" forecast would become 17,000, as it now reflects both the underlying residual forecast and the seasonal basis.

For trends such as growth and decline, the difference parameter $\boldsymbol{d}$ of ARIMA automatically attempts to smooth these by applying differencing to the series that it is modeling. With $\boldsymbol{d}$ = *1*, this process creates a new time series that, at each time step, consists of the original value minus its previous value. The first value in the series is eliminated because it doesn't have a previous value. To observe how this can smooth out trend, consider a time series with a linear growth trend: {*2, 4, 6, 8, 10*}. Differencing this series would create a new stationarity series with constant mean and variance: {*2, 2, 2, 2*}, because *4 - 2 = 6 - 4 = 8 - 6 = 10 - 8 = 2.*

A different set of preprocessing measures was applied to the raw time series prior to training with RNNs. Neural networks excel at capturing complex relationships between features (in this case historical data), therefore neither trend nor seasonality removal was performed. Instead each series was log-transformed and standardized to zero mean and unit variance. These steps were necessary for implementation of the algorithm detailed in Section 3.2, where the justification for applying these steps will be returned to as well. During preprocessing, each series' scikit-learn scaler used for standardizing was kept so that its stored parameters could later be used to inverse transform the forecasts. Following inverse transforming, the exponential function was applied to revert the original log-transform and bring the forecasted values back to a meaningful and interpretable scale.

## 3.2 Implementation

Implementing ARIMA in statsmodels, a Python library, was a relatively straightforward process as it is offered there more or less out-of-the-box. All that is needed is appropriately estimating values to use for $\{\boldsymbol{p},\ \boldsymbol{d},\ \boldsymbol{q}\}$. After the optimal lag parameter $\boldsymbol{p}$ of 12 was determined using trial and error and autocorrelation plots, the performance of various ARIMA configurations were compared, altering the differencing and moving average parameters $\boldsymbol{d}$ and $\boldsymbol{q}$ with each run. Each model iteration was fitted to each deseasoned series before being asked to forecast the final 12 months. Upon obtaining the raw forecast values, the pre-processing steps were reversed through returning seasonality and trend. Output was compared to the ground truth values with MAPE to identify parameters $\{\boldsymbol{p},\ \boldsymbol{d},\ \boldsymbol{q}\}$ that produce optimal forecasts.

It took longer, and was a much more demanding task, on the other hand to get RNNs to work with the time series available for this project, although it was moderately successful in the end. Having such data at the monthly granularity helps to smooth out day-to-day noise, but one issue that it presents involves a shortage of data, with each series only containing 174 values. On top of that, the goal was to predict 12 months into the future, which is a considerably more complex task than predicting one month. Experts in the field often object that deep learning models require thousands or tens of thousands of training examples to successfully predict even single values, so finding ways to overcome this perception was challenging. The solution required getting creative with the available data, and ultimately was motivated by successes in transfer learning - where a model can learn to make predictions in a small dataset after being trained on similar, but larger, datasets.

In review, three, "gold standard", time series from the commercial aviation industry were selected for this particular project - AA-DFW, DL-ATL, and UA-ORD, as the airports in these combinations serve as airline hubs for the airlines that they are paired with here. In the larger dataset however, there are a total of 80 time series for combinations that have either American, Delta, or United as the airline. A few examples include AA-DEN (American-Denver), DL-SEA (Delta-Seattle), and UA-LAX (United-Los Angeles). An insight that occurred is that the RNN could feasibly learn strong parameters for related time series (training series) by training on them, before being asked to predict the three series of interest (test series). Since the other time series come from the same industry and relate to the same airlines, they should be subject to the same structural forces of trend and seasonality. This approach is fundamentally different from the usual one taken in machine learning in that the model here has to transfer what it learns from the data that it sees to new data that is at best similar in structure, but not scale.

To translate this idea into a practical training algorithm, the RNN is run at various amounts of epochs over the 77 remaining training series. There are 77 instead of 80 because the three test series are removed from the training set to avoid leakage, which would otherwise violate many assumptions upon which valid machine learning practice is based (e.g. no sharing of data between training and test phases). At each epoch, the model randomly samples a certain number of series from the set, as determined by the batch size. In the process, it normalizes them with the steps described in Section 3.1 to bring them all to the same scale. After forward-propagating the batch through both encoder and decoder, backpropagation is run to update all weight parameters. As such, the number of parameter updates is determined entirely by the number of epochs. For every 100 training epochs, the model samples a fixed subset of time series to generate running validation scores. In this project, the data processing pipeline (loading, sampling, preprocessing, splitting, etc.) was implemented with custom Python functions - held in the utils.py file. These were in turn plugged into training and testing ops in Tensorflow, which provides library support for Seq2Seq RNNs. The Tensorflow models were implemented on a CPU, which is not ideal considering the cost of training time, leaving room for further experimentation and improvement when trained using GPUs.

### 3.3 Refinement

The initial ARIMA model that was fitted to the series of interests had parameters $\{\boldsymbol{p} = \mathit{12}, \boldsymbol{d} = \mathit{1}, \boldsymbol{q} = \mathit{1}\}$. Lag parameter $\boldsymbol{p}$ was chosen in accordance with the autocorrelation plots in Figure 4 to 6. With $\boldsymbol{q}$ fixed at 1, increasing difference parameter $\boldsymbol{d}$ beyond 2 caused instability and failure to converge, so little trial and error was necessary. The performance of $\boldsymbol{d}$ = 1 outperformed $\boldsymbol{d}$ = 2 for each of the three time series - perhaps because some stationary was already induced through removing seasonality. A small range of $\boldsymbol{q}$ was attempted in order to introduce a small stochastic element into the model. Among values 1, 2, 3, and 4, $\boldsymbol{q}$ = 2 performed best when combined with the lag and difference parameters above. In summary, the optimal ARIMA model was obtained with parameters $\{\boldsymbol{p} = \mathit{12}, \boldsymbol{d} = \mathit{1}, \boldsymbol{q} = \mathit{2}\}$. Using this model configuration, fitting and prediction was run one by one for time series AA-DFW, DL-ATL, and UA-ORD across the four dimensinos, with the results to follow in Section 4.1.

RNNs involve comparatively more tunable parameters than ARIMA, therefore exhaustively trying all their combinations was not feasible for this project - especially only using a CPU. Fortunately, no particular hyperparameter was identified that could significantly sway performance. Overall results seemed more bounded by the noisy nature of the data and limitations of the model architecture more than anything. Before exploring the process of hyperparameter searching, it must be noted that the RNNs model parameters were not optimized on an industry dimension basis, but instead over all four dimensions at the same time. This was done in order to encourage generalizeability, and so that there wouldn't be four separate sets of hyperparameters to keep track of. In other words, there was only one initial and one final model set of model parameters, not one for each dimension (passengers, flights, ASM, RPM).

An initial configuration with one GRU layer, five hidden dimensions, five epochs, batch size of three, a basic gradient descent optimizer, and L2 loss, was implemented in Tensorflow to validate the model architecture and ensure that inputs could successfully flow through and result in reasonable outputs. With this step achieved, the performance of several state-of-the-art optimizers based on stochastic gradient descent were compared, including AdaGrad, RMSProp, and Adam. RMSProp consistently achieved the superior performance, and therefore was included in the final model. L2 loss was chosen as the loss function to minimize, primarily because it is one of the only regression loss functions that comes out-of-the-box in Tensorflow. To mitigate overfitting, a regularization term was instead added to the L2 loss during training. Dropout was not used as the final RNN employed only two stacked GRU layers, leaving little space for this form of regularization. After experimenting with a wide variety of hidden layer and dimension combinations and comparing their MAPE results, the final model was implemented with hidden dimension size of 100 and two GRU layers. In general, it is common for RNNs to be wide, but not particularly deep, unlike convolutional neural networks where networks ten layers deep can still be considered shallow. 1000 epochs was found to be roughly the point were further gains in validation accuracy could not be made and a training batch size of three was kept from the beginning since the size of both the validation set and testing set was both three time series.

## 4 Results

### 4.1 Model Evaluation and Validation

Given the data and the task at hand, the final ARIMA model aligns well with expectations set forth in the project proposal and the problem statement. It does succeed in capturing how each time series develops over 12 months to a strong enough to degree to exceed the median model's performance. Furthermore, it succeeds in answering the problem statement, which asked whether or not time series in this industry could be predicted using only previous values from the same series. In this case, ARIMA was only fitted to a transformed version of each time series to predict values within that series. The parameters $\{\boldsymbol{p} = 12, \boldsymbol{d} = 1, \boldsymbol{q} = 2\}$ make sense because yearly patterns repeat in a seasonal time series (hence lagging 12 steps), the data is only moderately non-stationary after removing seasonality (hence a differencing factor of 1), and there's a small degree of error each series (hence an error model of 2).

Although the results of the Seq2Seq RNN approach deserve respect at first glance, the way that the model was trained does not exactly answer the question in the problem statement. In order to properly train this model, data had to be brought in from other related time series so that robust weight parameters could be learned for the prediction tasks across dimensions. Without being able to train on similar data, it's unlikely that a given series of length 174 would provide enough data for an RNN to converge on the weights necessary for good forecasts. In short, while the final model is reasonable, it was not expected at the problem outset.

Due to the random sampling step in the training algorithm, results for each run on a given dimension do vary, but not to the extent of breaking down the soundness of the model or the results. In fact, one could argue that the ability of the model to perform well consistently in the face of random training data (albeit sampled from the same distribution) actually further supports its robustness. The best average MAPE across the three airline-airport combinations is displayed for each dimension in Table 6. More often than not however, the RNN achieved strong performance on one airline-airport combination (1% to 2% MAPE), decent performance on another (3% to 5% MAPE), and poor performance on the last (6%+ MAPE). These relative performance differences likely correspond with the relative frequencies of airline series fetched at training. For example, if in a given run, training sampled 50% American series, 30% Delta series, and 20% United series, then it would make sense that the results on the American forecasts would exceed those on Delta and United. With this said, the results for the majority of runs only differed from the Table 6 results by 2 to 3 percentage points, so all in all the Seq2Seq RNN can be trusted to produce consistent results when the training data is altered slightly.

## 4.2 Justification

The results in Tables 5 and 6 demonstrate that both ARIMA and RNNs easily outperform the naive median model for this task - with the top scores in each dimension roughly 30 to 50% greater than the benchmark. Seq2Seq RNNs perform the best on average in each industry dimension, with the top MAPE scores highlighted in bold. Although ARIMA performs impressively in certain sections of the data, especially on DL-ATL, the Seq2Seq models were robust enough to make significant improvements in areas where ARIMA came up weak, such as UA-ORD.

With average absolute percentage errors in the to 2 to 4% range, these results are outstanding, however it must be reiterated that the particular time series used in this project were some of the cleanest in the dataset. Subjecting the final model to noisier time series that contain missing data, rocky trends, as well as sharp spikes and dips, would be a challenge to take up in future research. Overall, while the results successfully demonstrate a proof-of-concept for Seq2Seq RNNs applied to time series data, the problem is still far from being solved. To more conclusively address the problem statement regarding the predictive capacity of RNNs, a further test they should undergo is forecasting 12 months of truly future data. This could be forecasting Delta's passengers at Atlanta from Feb. 2018 to Jan. 2019 for example. In such a scenario, future data for related series would not be available either, therefore the model would have to be trained entirely on historical series, which was not exactly the case in this project.

| Dimension | American-Dallas | Delta-Atlanta | United-Chicago | Avg. |
|---|---|---|---|---|
| Passengers | 3.076 | 1.816 | 7.922 | 4.271 |
| Flights | 2.839 | 3.302 | 11.143 | 5.761 |
| ASM | 2.133 | 1.113 | 6.330 | 3.192 |
| RPM | 6.274 | 1.705 | 5.273 | 4.417 |

Table 5: MAPE results for ARIMA

| Dimension | American-Dallas | Delta-Atlanta | United-Chicago | Avg. |
|---|---|---|---|---|
| Passengers | 3.743 | 3.513 | 4.122 | **3.793** |
| Flights | 5.909 | 2.887 | 3.603 | **4.133** |
| ASM | 3.418 | 1.739 | 3.013 | **2.723*** |
| RPM | 4.678 | 2.980 | 2.523 | **3.394** |

Table 6: MAPE results for Seq2Seq RNNs

## 5 Conclusion

### 5.1 Free-Form Visualization

Two figures here display visual examples of the manner in which ARIMA and RNNs make their forecasts. Figure 9 displays the plot of ARIMA's best score forecasting ASM for AA-DFW. Figure 10 shows the results of a random iteration of the Seq2Seq RNN over this same time series. It's surprising how well both models were able to capture the seasonality in the data - adjusting the forecasts up and down in sync with the annual cycle. What's most striking about both models is the final prediction (March 2017), which turns out to be nearly perfect and by far the best overall. It's possible that by having March 2016 as the last value in the training set, this value was most "fresh" at forecasting time, especially considering than an RNN does build its internal state progressively over time steps. Another interesting aspect is that both ARIMA and the RNN tended to overestimate on this particular time series - ARIMA 11 out of 12 times and RNN 12 out of 12. This perhaps indicates that features inherent to the individual time series are more influential in limiting the accuracy forecasts, rather than features of the models themselves. All in all, both models handled the time series well, although this wasn't the best run for the RNN when considering the average over the four dimensions. Table 6 contains the best average performances over many successive trials.

### 5.2 Reflection

In summary, this project compared the performance of two modern time series forecasting techniques - ARIMA and RNNs - on data from the commercial aviation industry that was scraped from the U.S. Department of Transportation's publicly available sources [10]. For testing, three "gold-standard" time series datasets - American-Dallas/Fort-Worth, Delta-Atlanta, and United-Chicago - were selected due to their completeness and quality. Within each dataset were four industry dimensions - passengers, flights, available seat-miles, and revenue passenger-miles. Each series contained evidence of trend and seasonality, but no sharp spikes or dips, as is common in other series in the dataset. The dimensions were modeled separately with shared sets of model parameters for both models. The central task

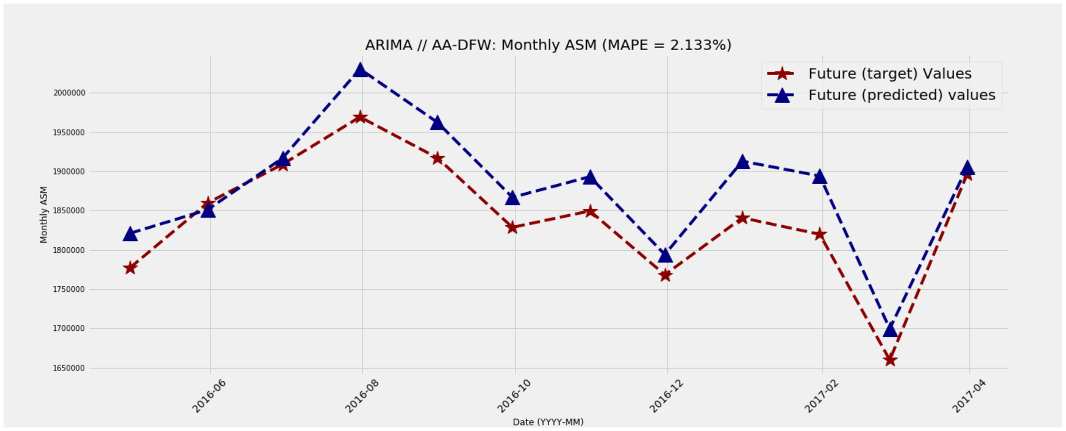

Figure 9: ARIMA: Ground truth vs. forecast (ASM for AA-DFW)

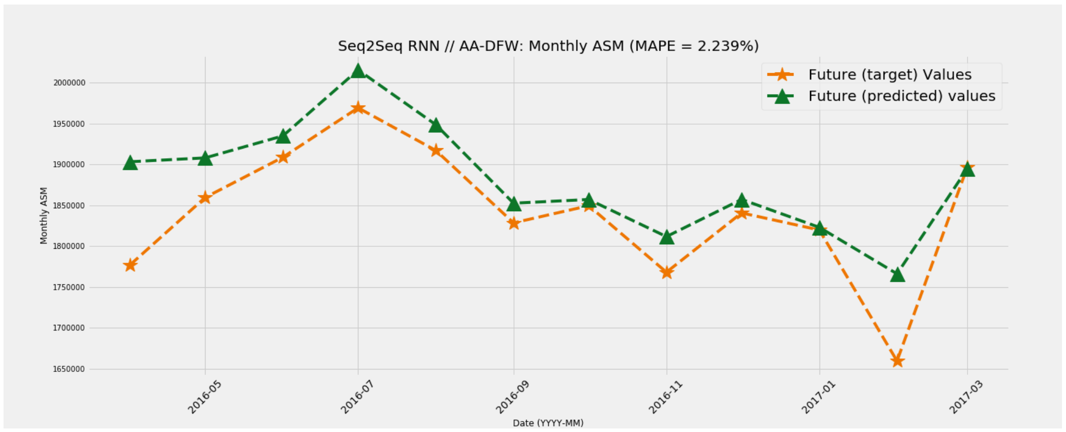

Figure 10: Seq2Seq RNNs: Ground truth vs. forecast (ASM for AA-DFW)

was to make forecasts for these dimensions 12 months into the future (April 2016 to March 2017) only using values for the previous 162 months (October 2002 to March 2016).

After bringing stationarity to each series through removing seasonality, ARIMA was fitted to the data with various model parameters. The best model outperformed a naive median model by 20 to 30% across the various dimensions, as measured by mean absolute percentage error. In order to train RNNs in the face of external data, a training algorithm was designed and implemented that randomly sampled related time series from the larger dataset, normalized their scales, and fed them into the model in batches. Upon training an optimized Seq2Seq RNN for 1000 epochs, MAPE scores of 30 to 50% greater than the naive median model were achieved.

The most difficult step in this project that also produced the most interesting result was designing an RNN model architecture and training algorithm that could overcome both problems of data scarcity and multiple outputs expectation. Over the course of many months, several approaches such as those in [12] and [13] were attempted on the time series here, but the results were distressing on multiple outputs - the naive median model performed much better. The project was nearly abandoned when a Kaggle competition focusing on forecasting challenging Wikipedia time series data wrapped up. With the winner of the competition generous enough to reveal both the code and explanation of his Seq2Seq RNN [11], an architecture with hope was discovered. After combining it with the random sampling algorithm discussed in Section 3.2, results that were worth reporting were finally produced. A major limitation of the final RNN model is that it had to train on data that it would not necessarily have access to if the task was to predict real future values. For example, the model here may have trained on the entire length 174 Delta-Seattle series in order to predict the final 12 values for Delta-Atlanta. When predicting true future values, the model would not have had access to the entire Delta-Seattle series, but only the first 162 values. It's possible that the same model used in this project could match the performance shown in Table 6 even with a different time series segmenting scheme, but that's again an avenue for future research. The results in [11] do grant us hope though, in that they were produced entirely from models trained on historical data being used to predict truly unseen future values. Though the research is young, it's apparent that deep learning sequence architectures such as Seq2Seq RNNs have vast potential as candidates for time series modeling and forecasting.

## 5.3 Improvement

Although much more difficult time series data could be supplied to test the performance of the final RNNs in this project, there are a variety of ways to ensure their robustness keeps up. Having access to one or more GPUs to more efficiently test various model architectures and parameter configurations is a small step towards a stronger model. Furthermore, univariate time series was used in this project, but its not hard to see how including additional time series such as GDP, employment, inflation, etc. (multivariate time series) could improve performance. Including research with these series was in the original plan in the proposal, but the decision was made to leave them out to control the project scope.

The strongest area for improvement, however lies in the vast array of modifications and enhancements that can be made the basic Seq2Seq RNN - that researchers in the field are inventing day-to-day. The winning Kaggle model in [11] implemented an attention interface on top of the basic Seq2Seq setup, that allows the decoder to ”focus” on various aspects of the incoming data as it unpacks the encoder state across a number of time steps. A model trained as such could feasibly have the ability to learn where the most relevant pieces of historical data lie when predicting on a point-by-point basis. For example, when predicting July 2016, it could use ”knowledge” of seasonality to know that it needs to look to July 2015 and then project one year forward. A simple diagram that illustrates the concept is shown in Figure 11. Considering the results achieved by this kind of model in the Kaggle competition on time series, there's confidently superior models that could produce significantly better scores than the basic Seq2Seq RNN implemented in this project, and so there is still much work to be done.

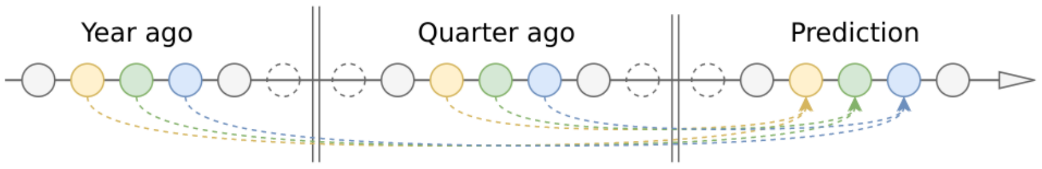

Figure 11: Attention interface for improving deep time series models (`https://github.com/Arturus/kaggle-web-traffic/blob/master/how_it_works.md`)

## 5.4 References

[1] Aas, K., & Dimakos, X. K. (2004). Statistical modelling of financial time series. Norwegian Computing Center. Retrieved from `https://www.nr.no/files/samba/bff/SAMBA0804.pdf`

[2] Adebiyi, A. A., Adewumi, A. O., & Ayo, C. K. (2014). Stock Price Prediction Using the ARIMA Model. International Conference on Computer Modelling and Simulation 2014. Retrieved from `http://ijssst.info/Vol-15/No-4/data/4923a105.pdf`.

[3] Falode, O., & Udomboso, C. (2016). Predictive Modeling of Gas Production, Utilization and Flaring in Nigeria using TSRM and TSNN: A Comparative Approach. Open Journal of Statistics, 6(1), 194-207. Retrieved from `http://www.scirp.org/journal/PaperInformation.aspx?PaperID=63994`.

[4] Gers, F. A., Eck, D., & Schmidhuber, J. (2001). Applying LSTM to time series predictable through time-window approaches. International Conference on Artificial Networks 2001 (pp. 669-676). Retrieved from `https://link.springer.com/chapter/10.1007/3-540-44668-0_93`.

[5] Kohzadi, N., Boyd, M. S., Kermanshahi, B., & Kaastra, I. (1996). A comparison of artificial neural network and time series models for forecasting commodity prices. Neurocomputing, 10(2), 169-181. Retrieved from `http://www.sciencedirect.com/science/`

article/pii/0925231295000208.

[6] Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2016). Learning to diagnose with LSTM Recurrrent Neural Networks. International Conference on Learning Representations 2016. Retrieved from https://arxiv.org/abs/1511.03677.

[7] Voyant, C., Nivet, M. L., Paoli, C., Muselli, M., & Notton, G. (2014). Meteorological time series forecasting based on MLP modelling using heterogeneous transfer functions. International Conference on Mathematical Modeling in Physical Sciences 2014. Retrieved from https://arxiv.org/abs/1404.7255.

[8] Dickey, D. A.; Fuller, W. A. (1979). Distribution of the estimators for autoregressive time series with a unit root. Journal of the American Statistical Association. 74(366), 427–431. JSTOR 2286348. doi:10.2307/2286348.

[9] Jozefowicz, R., Zaremba, W., Sutskever, I. (2015). An Empirical Exploration of Recurrent Network Architectures. Proceedings of the 32nd International Conference on Machine Learning, 37. Retrieved from http://proceedings.mlr.press/v37/jozefowicz15.pdf

[10] https://www.transtats.bts.gov/Data_Elements.aspx

[11] https://github.com/Arturus/kaggle-web-traffic

[12] https://github.com/jaungiers/LSTM%2DNeural%2DNetwork%2Dfor%2DTime%2DSeries%2DPrediction

[13] https://machinelearningmastery.com/time%2Dseries%2Dforecasting%2Dlong%2Dshort%2Dterm%2Dmemory%2Dnetwork%2Dpython