**Five Practical Applications of the LSTM Model for Time Series, with Code**

How to implement an advanced neural network model in several different time series contexts

[[Michael Keith](https://mikekeith52.medium.com/?source=post_page-----a7aac0aa85c0--------------------------------)](https://mikekeith52.medium.com/?source=post_page-----a7aac0aa85c0--------------------------------)

[[Towards Data Science](https://towardsdatascience.com/?source=post_page-----a7aac0aa85c0--------------------------------)](https://towardsdatascience.com/?source=post_page-----a7aac0aa85c0--------------------------------)

[Michael Keith](https://mikekeith52.medium.com/?source=post_page-----a7aac0aa85c0--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F85177a9cbd35&operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Ffive-practical-applications-of-the-lstm-model-for-time-series-with-code-a7aac0aa85c0&user=Michael+Keith&userId=85177a9cbd35&source=post_page-85177a9cbd35----a7aac0aa85c0---------------------post_header-----------)

Published in

[Towards Data Science](https://towardsdatascience.com/?source=post_page-----a7aac0aa85c0--------------------------------)

·

11 min read

·

Sep 22, 2023

151



Photo by [Andrew Svk](https://unsplash.com/@andrew_svk?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

When I wrote [Exploring the LSTM Neural Network Model for Time Series](https://medium.com/towards-data-science/exploring-the-lstm-neural-network-model-for-time-series-8b7685aa8cf) in January, 2022, my goal was to showcase how easily the advanced neural network could be implemented in Python using [scalecast](https://github.com/mikekeith52/scalecast), a time series library I developed to facilitate my own work and projects. I did not think that it would be viewed over 10s of thousands of times and appear as the first hit on Google when searching “lstm forecasting python” for over a year after I published it (when I checked today, it was still number two).

I haven’t tried to call much attention to that article because I never thought, and still don’t think, it is very good. It was never meant to be a guide on the best way to implement the LSTM model, but rather a simple exploration of its utility for time series forecasting. I tried to answer such questions as: what happens when you run the model with default parameters, what happens when you adjust its parameters in this way or that, how easily can it be beat by other models on certain datasets, etc. However, judging by the blog posts, Kaggle notebooks, and even the [Udemy course](https://www.udemy.com/course/uniform-ml-dl/) that I keep seeing pop up with the code from that article copied verbatim, it’s clear many people were taking the piece for the former value, not the latter. I understand now that I did not clearly lay out my intentions.

Today, to expand upon that article, I want to showcase how one should apply the LSTM neural network model, or at least how I would apply it, to fully realize its value for time series forecasting problems. Since I wrote the first article, we have been able to add many new and innovative features to the scalecast library that make using the LSTM model much more seamless and I will take this space to explore some of my favorites. There are five applications for LSTM that I think will all work fantastically using the library: **univariate forecasting, multivariate forecasting, probabilistic forecasting, dynamic probabilistic forecasting, and transfer learning.**

Before starting, be sure to run on terminal or command line:

pip install --upgrade scalecast

The complete notebook developed for this article is located [here.](https://github.com/mikekeith52/scalecast-examples/blob/main/lstm/lstm_latest.ipynb)

One final note: in each example, I may use the terms “RNN” and “LSTM” interchangeably. Alternatively, RNN may be displayed on a given graph of an LSTM forecast. The long short-term memory (LSTM) neural network is a type of recurrent neural network (RNN), with additional memory-related parameters. In scalecast, the rnn model class can be used to fit both simple RNN and LSTM cells in models ported from [tensorflow](https://www.tensorflow.org/).

**1. Univariate forecasting**

The most common and most obvious way to use the LSTM model is when doing a simple univariate forecasting problem. Although the model fits many parameters that should make it sophisticated enough to learn trends, seasonality, and short-term dynamics in any given time series effectively, I have found that it does much better with stationary data (data that doesn’t exhibit trends or seasonality). So, with the air passengers dataset — which is available on [Kaggle](https://www.kaggle.com/rakannimer/air-passengers) with an Open Database license — we can easily create an accurate and reliable forecast using fairly simple hyperparameters, if we simply detrend and de-season the data:

transformer = Transformer(  
 transformers = [  
 ('DetrendTransform',{'poly\_order':2}),  
 'DeseasonTransform',  
 ],  
)

We also want to make sure to revert the results to their original level when we are done:

reverter = Reverter(  
 reverters = [  
 'DeseasonRevert',  
 'DetrendRevert',  
 ],  
 base\_transformer = transformer,  
)

Now, we can specify the network parameters. For this example, we will use 18 lags, one layer, a tanh activation function, and 200 epochs. Feel free to explore your own, better parameters!

def forecaster(f):  
 f.set\_estimator('rnn')  
 f.manual\_forecast(  
 lags = 18,  
 layers\_struct = [  
 ('LSTM',{'units':36,'activation':'tanh'}),  
 ],  
 epochs=200,  
 call\_me = 'lstm',  
 )

Combine everything into a pipeline, run the model, and view the results visually:

pipeline = Pipeline(  
 steps = [  
 ('Transform',transformer),  
 ('Forecast',forecaster),  
 ('Revert',reverter),  
 ]  
)  
  
f = pipeline.fit\_predict(f)  
  
f.plot()  
plt.show()

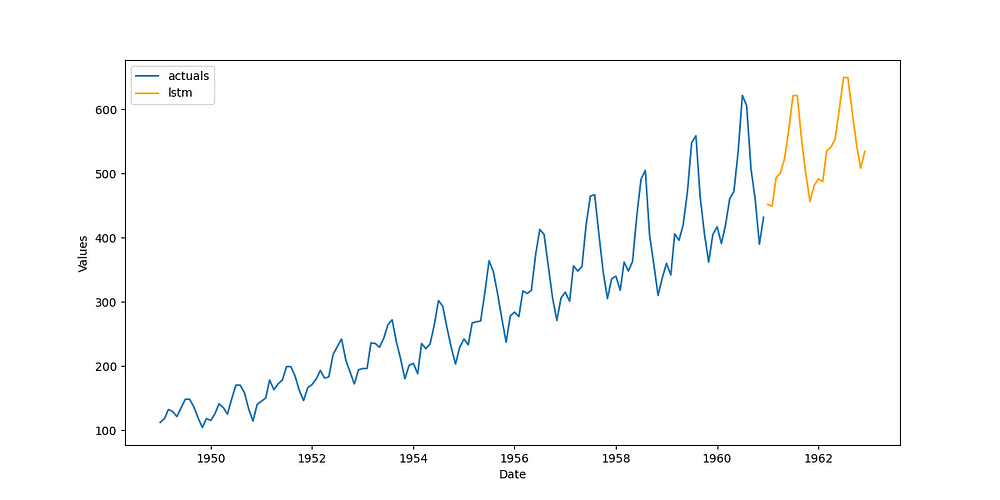


Image by author

Good enough and much better than anything I demonstrated in the other article. To extend this application, you can try using different lag orders, adding seasonality to the model in the form of Fourier terms, finding better series transformations, and tuning the model hyperparameters with cross-validation. Some of how to do this will be demonstrated in the subsequent sections.

**2. Multivariate forecasting**

Let’s say that we have two series that we expect move together. We can create an LSTM model that takes both series into consideration when making predictions with the hope that we improve the model’s overall accuracy. This is, of course, multivariate forecasting.

For this example, I will use the Avocados dataset, available on [Kaggle](https://www.kaggle.com/datasets/neuromusic/avocado-prices) with an Open Database license. It measures the price and quantity sold of avocados on a weekly level over different regions of the United States. We know from economic theory that price and demand are closely interrelated, so using price as a leading indicator, we might be able to more accurately forecast the amount of avocados sold than just by using historical demand in a univariate context.

The first thing we will do is transform each series. We can search for an “optimal” set of transformations (meaning transformations that are scored out-of-sample) by running the following code:

data = pd.read\_csv('avocado.csv')  
  
# demand  
vol = data.groupby('Date')['Total Volume'].sum()  
# price  
price = data.groupby('Date')['AveragePrice'].sum()  
  
fvol = Forecaster(  
 y = vol,  
 current\_dates = vol.index,  
 test\_length = 13,  
 validation\_length = 13,  
 future\_dates = 13,  
 metrics = ['rmse','r2'],  
)  
  
transformer, reverter = find\_optimal\_transformation(  
 fvol,  
 set\_aside\_test\_set=True, # prevents leakage so we can benchmark the resulting models fairly  
 return\_train\_only = True, # prevents leakage so we can benchmark the resulting models fairly  
 verbose=True,  
 detrend\_kwargs=[  
 {'loess':True},  
 {'poly\_order':1},  
 {'ln\_trend':True},  
 ],  
 m = 52, # what makes one seasonal cycle?  
 test\_length = 4,  
)

The recommended transformation from this process is a seasonal adjustment, assuming 52 periods makes one season, as well as a [robust scale](https://scalecast.readthedocs.io/en/latest/Forecaster/SeriesTransformer.html#src.scalecast.SeriesTransformer.SeriesTransformer.RobustScaleTransform) (scaling that is robust to outliers). We can then fit that transformation on the series and call a univariate LSTM model to benchmark the multivariate model against. This time, we will use a hyperparameter-tuning process by generating a grid of possible activation functions, layer sizes, and dropout values:

rnn\_grid = gen\_rnn\_grid(  
 layer\_tries = 10,  
 min\_layer\_size = 3,  
 max\_layer\_size = 5,  
 units\_pool = [100],  
 epochs = [25,50],  
 dropout\_pool = [0,0.05],  
 callbacks=EarlyStopping(  
 monitor='val\_loss',  
 patience=3,  
 ),  
 random\_seed = 20,  
) # creates a grid of hyperparameter values to tune the LSTM model

This function gives a good way to ingest a manageable grid into our object but also have enough randomness to have a good candidates of parameters to choose from. Now we fit the univariate model:

fvol.add\_ar\_terms(13) # the model will use 13 series lags  
fvol.set\_estimator('rnn')  
fvol.ingest\_grid(rnn\_grid)  
fvol.tune() # uses a 13-period validation set  
fvol.auto\_forecast(call\_me='lstm\_univariate')

To extend this into a multivariate context, we can transform the price time series with the same set of transformations that we used on the other series. Then, we ingest 13 price lags into the Forecaster object and fit a new LSTM model:

fprice = Forecaster(  
 y = price,  
 current\_dates = price.index,  
 future\_dates = 13,  
)  
  
fprice = transformer.fit\_transform(fprice)  
  
fvol.add\_series(fprice.y,called='price')  
fvol.add\_lagged\_terms('price',lags=13,drop=True)  
fvol.ingest\_grid(rnn\_grid)  
fvol.tune()  
fvol.auto\_forecast(call\_me='lstm\_multivariate')

We can also benchmark a naïve model and plot the results at the original series level, along with the out-of-sample test set:

# naive forecast for benchmarking  
fvol.set\_estimator('naive')  
fvol.manual\_forecast()  
  
fvol = reverter.fit\_transform(fvol)  
  
fvol.plot\_test\_set(order\_by='TestSetRMSE')  
plt.show()

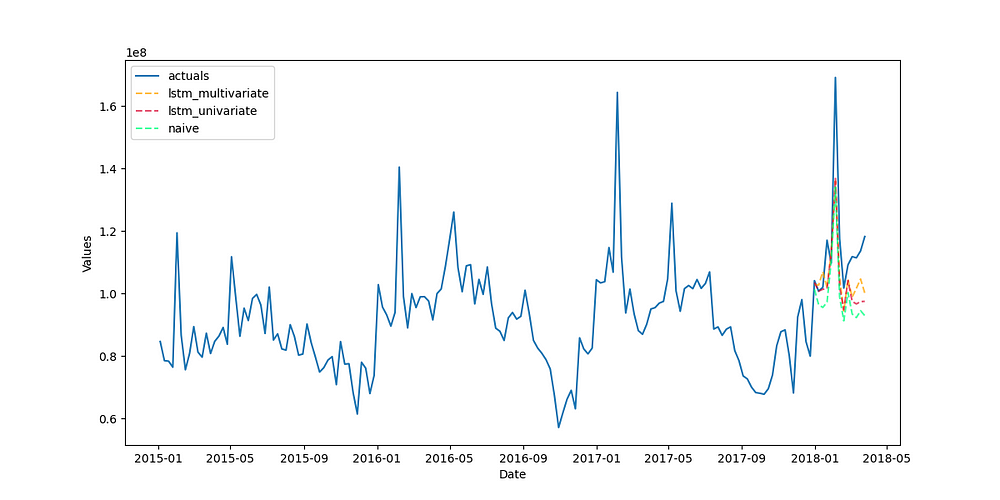


Image by author

Judging by how all three models clustered together visually, what led to most of the accuracy on this particular series were the applied transformations — that’s how the naïve model ended up so comparable to both the LSTM models. Still, the LSTM models are an improvement, with the multivariate model scoring and r-squared of 38.37% and the univariate mode 26.35%, compared to the baseline of -6.46%.

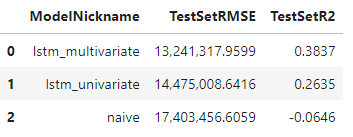


Image by author

One thing that might have hindered the LSTM models from performing better on this series is how short it is. With only 169 observations, that may not be enough history for the model to sufficiently learn the patterns. However, any improvement over some naïve or simple model can be considered a success.

**3. Probabilistic forecasting**

Probabilistic forecasting refers to the ability of a model to not only make point predictions, but to provide estimates of how far off in either direction the predictions are likely to be. Probabilistic forecasting is akin to forecasting with [confidence intervals](https://en.wikipedia.org/wiki/Confidence_interval), a concept that has been around for a long time. A quickly emerging way to produce probabilistic forecasts is by applying a [conformal confidence interval](https://en.wikipedia.org/wiki/Conformal_prediction) to the model, using a calibration set to determine the likely dispersion of the actual future points. This approach has the advantage of being applicable to any machine learning model, regardless of any assumptions that model makes about the distribution of its inputs or residuals. It also provides certain coverage guarantees that are extremely useful to any ML practitioner. We can apply the conformal confidence interval to the LSTM model to produce probabilistic forecasts.

For this example, we will use the monthly housings starts dataset available on [FRED](https://fred.stlouisfed.org/series/HOUSTNSA), an open-source database of economic time series. I will use data from January, 1959 through December, 2022 (768 observations). First, we will once again search for the optimal set of transformations, but this time using an LSTM model with 10 epochs to score each transformation try:

transformer, reverter = find\_optimal\_transformation(  
 f,  
 estimator = 'lstm',  
 epochs = 10,  
 set\_aside\_test\_set=True, # prevents leakage so we can benchmark the resulting models fairly  
 return\_train\_only = True, # prevents leakage so we can benchmark the resulting models fairly  
 verbose=True,  
 m = 52, # what makes one seasonal cycle?  
 test\_length = 24,  
 num\_test\_sets = 3,  
 space\_between\_sets = 12,  
 detrend\_kwargs=[  
 {'loess':True},  
 {'poly\_order':1},  
 {'ln\_trend':True},  
 ],  
)

We will randomly generate a hyperparameter grid again, but this time we can make its search space very big, then limit it manually to 10 tries when we the model is fit later so that we can cross validate the parameters in a reasonable amount of time:

rnn\_grid = gen\_rnn\_grid(  
 layer\_tries = 100,  
 min\_layer\_size = 1,  
 max\_layer\_size = 5,  
 units\_pool = [100],  
 epochs = [100],  
 dropout\_pool = [0,0.05],  
 validation\_split=.2,  
 callbacks=EarlyStopping(  
 monitor='val\_loss',  
 patience=3,  
 ),  
 random\_seed = 20,  
) # make a really big grid and limit it manually

Now we can build and fit the pipeline:

def forecaster(f,grid):  
 f.auto\_Xvar\_select(  
 try\_trend=False,  
 try\_seasonalities=False,  
 max\_ar=100  
 )  
 f.set\_estimator('rnn')  
 f.ingest\_grid(grid)  
 f.limit\_grid\_size(10) # randomly reduce the big grid to 10  
 f.cross\_validate(k=3,test\_length=24) # three-fold cross-validation  
 f.auto\_forecast()  
  
pipeline = Pipeline(  
 steps = [  
 ('Transform',transformer),  
 ('Forecast',forecaster),  
 ('Revert',reverter),  
 ]  
)  
  
f = pipeline.fit\_predict(f,grid=rnn\_grid)

Because we set aside a test-set of sufficient size in the Forecaster object, the results automatically give us the 90% probabilistic distributions for each point estimate:

f.plot(ci=True)  
plt.show()

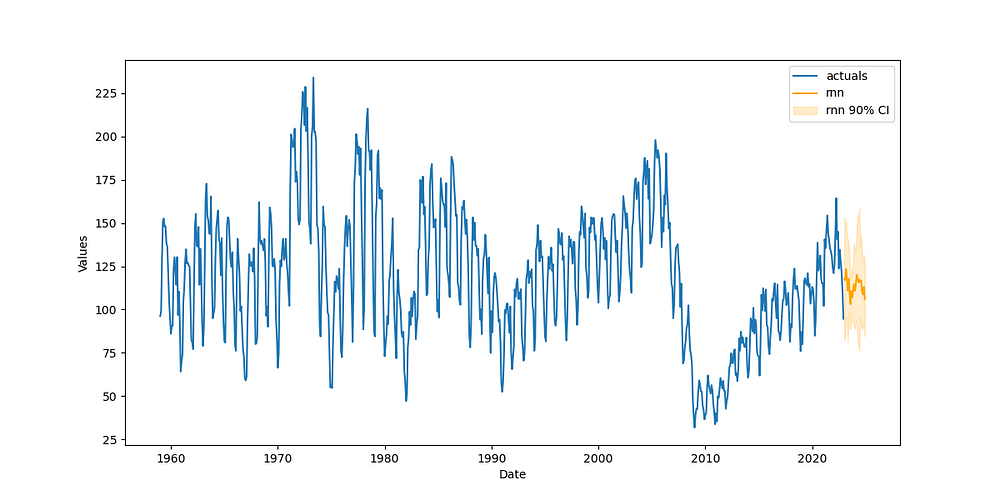


Image by author

**4. Dynamic probabilistic forecasting**

The previous example provided a static probabilistic prediction, where each upper and lower bound along the forecast is equally far away from the point estimate as any other upper and lower bound attached to any other point. When predicting the future, it is intuitive that the further out one attempts to forecast, the wider the error will disperse — a nuance not captured with the static interval. There is a way to achieve a more dynamic probabilistic forecast with the LSTM model by using backtesting.

Backtesting is the process of iteratively refitting the the model, predicting it over different forecast horizons, and testing its performance over each iteration. Let’s take the pipeline specified in the last example and backtest it 10 times. We need at least 10 backtest iterations to build confidence intervals at the 90% level:

backtest\_results = backtest\_for\_resid\_matrix(  
 f,  
 pipeline=pipeline,  
 alpha = .1,  
 jump\_back = 12,  
 params = f.best\_params,  
)  
  
backtest\_resid\_matrix = get\_backtest\_resid\_matrix(backtest\_results)

We can analyze the absolute values of the residuals over each iteration visually:

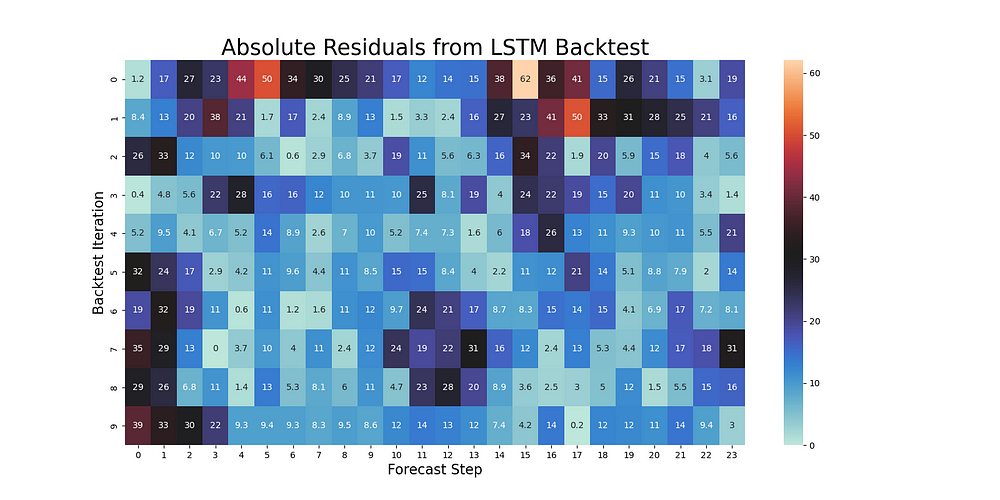


Image by author

What’s interesting about this particular example is that the largest errors are not usually on the last steps of the forecast, but actually over steps 14–17. This can happen with series that have odd seasonal patterns. The presence of outliers can also affect this pattern. Either way, we can use these results to now replace the static confidence intervals with dynamic intervals that are conformal along each step:

overwrite\_forecast\_intervals(  
 f,  
 backtest\_resid\_matrix=backtest\_resid\_matrix,  
 alpha=.1, # 90% intervals  
)  
f.plot(ci=True)  
plt.show()

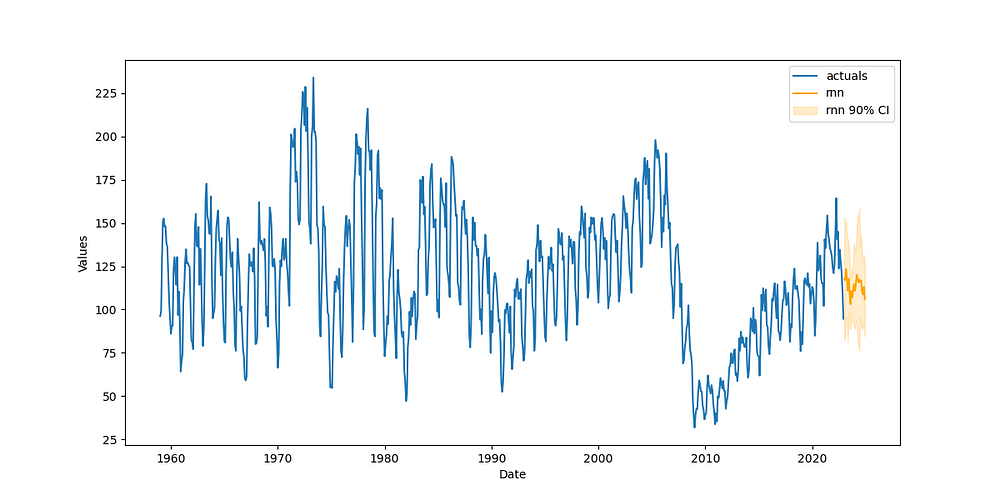


Image by author

**5. Transfer learning**

Transfer learning is useful when we wish to use a model outside of the context in which it was fit. There are two specific scenarios where I will demonstrate its utility: making predictions when new data in a given time series becomes available and making predictions on a related time series with similar trends and seasonality.

**Scenario 1: New data from the same series**

We can use the same housing dataset as in the previous two examples, but let’s say some time has passed and we now have data available through June, 2023.

df = pdr.get\_data\_fred(  
 'CANWSCNDW01STSAM',  
 start = '2010-01-01',  
 end = '2023-06-30',  
)  
  
f\_new = Forecaster(  
 y = df.iloc[:,0],  
 current\_dates = df.index,  
 future\_dates = 24, # 2-year forecast horizon  
)

We will remake our pipeline with the same transformations, but this time, use a transfer forecast instead of the normal scalecast forecast procedure that fits a model:

def transfer\_forecast(f\_new,transfer\_from):  
 f\_new = infer\_apply\_Xvar\_selection(infer\_from=transfer\_from,apply\_to=f\_new)  
 f\_new.transfer\_predict(transfer\_from=transfer\_from,model='rnn',model\_type='tf')  
  
pipeline\_can = Pipeline(  
 steps = [  
 ('Transform',transformer),  
 ('Transfer Forecast',transfer\_forecast),  
 ('Revert',reverter),  
 ]  
)  
  
f\_new = pipeline\_can.fit\_predict(f\_new,transfer\_from=f)

Even though the name of the relevant function is still fit\_predict(), there actually is no fitting and only predicting in the pipeline as it is written. This greatly reduces the amount of time we would have needed to refit and re-optimize the model. We then view the results:

f\_new.plot()  
plt.show('Housing Starts Forecast with Actuals Through June, 2023')  
plt.show()

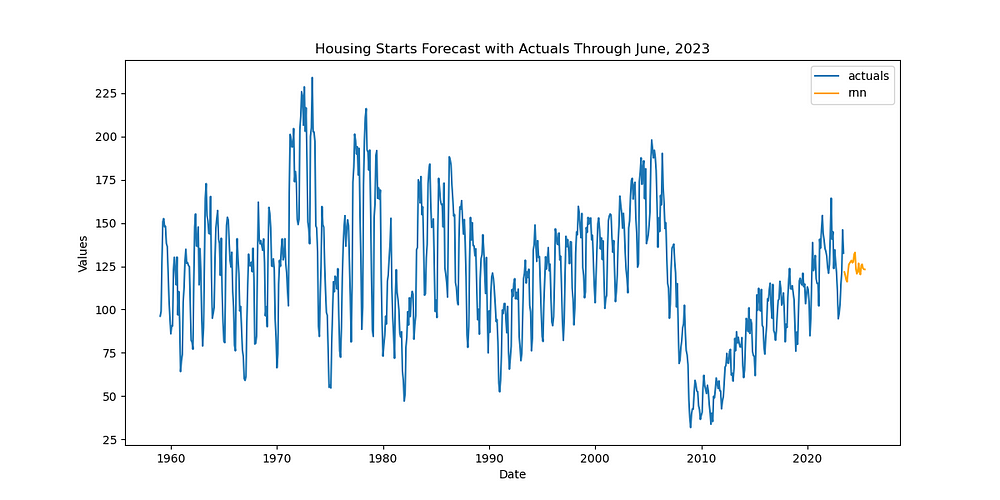


Image by author

**Scenario 2: A new time series with similar characteristics**

For the second scenario, we can use the hypothetical situation of wanting to use the model trained on the housing dynamics in the United States to predict housing starts in Canada. Disclaimer: I don’t know if this is actually a good idea — it is just one scenario I thought of to demonstrate how this would be done. But I imagine it could be useful and the code involved can be transferred to other situations (maybe for situations where you have short series that exhibit similar dynamics as a longer series that you have already fit a well-performing model to). In that case, the code is actually exactly the same as the **Scenario 1**code; the only difference is the data we load into the object:

df = pdr.get\_data\_fred(  
 'CANWSCNDW01STSAM',  
 start = '2010-01-01',  
 end = '2023-06-30',  
)  
  
f\_new = Forecaster(  
 y = df.iloc[:,0],  
 current\_dates = df.index,  
 future\_dates = 24, # 2-year forecast horizon  
)  
  
def transfer\_forecast(f\_new,transfer\_from):  
 f\_new = infer\_apply\_Xvar\_selection(infer\_from=transfer\_from,apply\_to=f\_new)  
 f\_new.transfer\_predict(transfer\_from=transfer\_from,model='rnn',model\_type='tf')  
  
pipeline\_can = Pipeline(  
 steps = [  
 ('Transform',transformer),  
 ('Transfer Forecast',transfer\_forecast),  
 ('Revert',reverter),  
 ]  
)  
  
f\_new = pipeline\_can.fit\_predict(f\_new,transfer\_from=f)  
  
f\_new.plot()  
plt.show('Candian Housing Starts Forecast')  
plt.show()

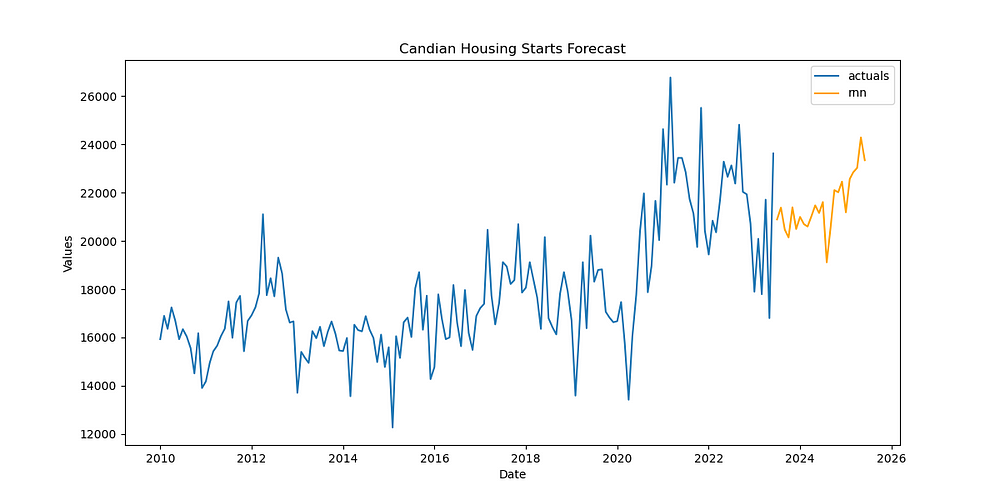


Image by author

I think the forecast looks believable enough for this to be an interesting application of LSTM transfer learning.

**Conclusion**

For many forecasting use cases, the LSTM model can be an interesting solution. In this post, I demonstrated how to apply the LSTM model for five different purposes with Python code. If you found it useful, give [scalecast a star on GitHub](https://github.com/mikekeith52/scalecast) and be sure to give me a follow here on Medium to be updated on the latest and greatest with the package. To provide feedback, constructive criticism or if you have questions about this code, feel free to email me: mikekeith52@gmail.com.