A Novel Application of Deep Learning on Time series forecasting

March 26, 2018 Hudson Zhao

There is no doubt on the importance of time series analysis. The applications of the techniques are manifold, included in Statistics, Mathematical Finance, Econometrics, Sales forecasting/Inventory studies, Signal processing, Weather forecasting/Earthquake prediction, Pattern recognition, and much more. The techniques have been long-established, for example, in both Statistics and Engineering literatures. In a very loose sense, I might split the techniques to two major frameworks, namely, Box-Jenkins and State Space Models even though there are overlaps between those two.

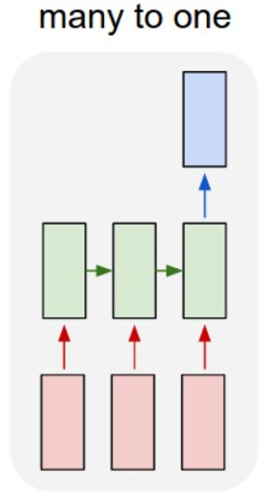
This is not an article to discuss and compare those two frameworks, even though my personal experiences might favor more to State Space Models, especially in long-term forecasting. Instead, the article more interests in how we can use machine learning methods to resolve those problems when sequential dependence and correlation exist in the data.

Machine learning(ML)/Artificial Intelligence(AI) have been attracting a huge amount of attentions from both academia and industry over many years. Deep Learning might be among one of those top “hypes”. Researchers and practitioners with different background, for example, Computer science, Engineering, and Mathematics, consistently push the envelope of the current development to new levels. There are so many amazing applications changing our vision on the technique. Some of the notable areas, but definitely not an exhaustive list, include Image/Video Segmentation/Object Detection, Image Caption, Natural Language Processing, Machine Translation, Robotics, and many others. However, to my very limited knowledge, most of the breakthroughs might mainly come from engineering perspectives. For example, those well-known convolutional nets, say AlexNet, GoogLeNet, ResNet, VGG, Inception, in image and video applications, focus mainly on the architecture of the networks. Encoder-decoder models and attention models used in Machine Translate literature also focus on the architecture of the networks to resolve the specific problems in this domain. On the other hand, the studies on the theoretical properties of the deep learning models are still open, and expect more breakthroughs.

Recurrent Neural Network (RNN) is one type of deep learning models. There are tons of tutorials out there, and some of them are really good. This article won’t duplicate those introductory material. However, briefly speaking, RNN handles the use case as “If I want to predict the next word in a sentence I better know which words came before it”. The temporal structure handled by RNN makes it a natural candidate for the data set with sequential dependence. Even though the output from the network memory cells are not exactly same as those previously observed data used in Box-Jenkins models or State Space Models, but it still could be a good proxy.

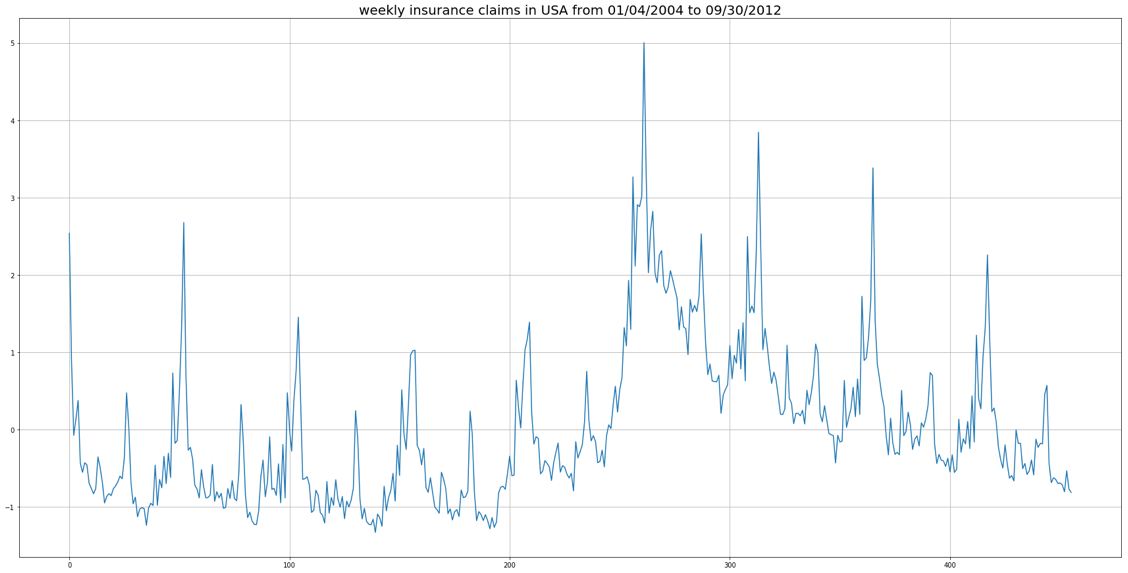
Using RNN or LSTM(Long short-term memory networks) for the time series analysis isn’t an innovation by this article either. There also exist many introductory material on how to use RNN/LSTM to do the time forecasting job. To my limited knowledge, many tech giants already used RNN/LSTM to resolve their time series related problems, naming a few of them, Uber, NASA (if readers are interested in those applications we can discuss more).

Sorry for a bit long preamble, let’s look at what this article is really about. In a very general sense, the way to use LSTM in most of the applications mentioned above in the sake of time series forecasting is similar to the way we use Box-Jenkins models or State Space Models. Let’s make it concrete. Suppose Yt. are the target variable of interest, then LSTM model aims to capture the dependence structure between Yt  with Yt-1 …… Yt-L, where *L* is the look-back length in the LSTM setting. The dependence structure in LSTM is usually in a non-linear fashion, and this is one of the major facts that makes LSTM differentiate with other linear models such as ARIMA or Dynamic Linear models. The following plot, from Andrej Karpathy’s awesome post, can depict this architecture, where the inputs of the networks, the pink boxes, are the previously observed Yt, and the blue one is the target of interest, the prediction at the current time t.



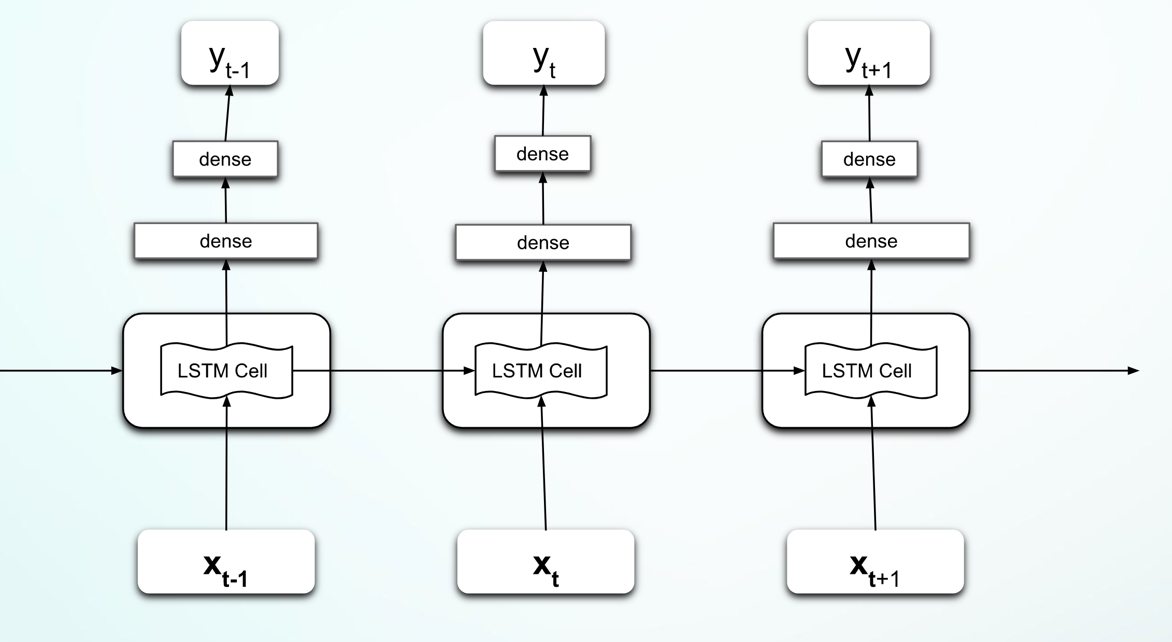
However the question considered by this article is slightly different. Suppose Yt, ……, Yt+L are not observed, but some exogenous variables, denoted as Xt, ……, Xt+L, are available, and we assume those un-observed Yt  are temporally dependent, and those Xt as well. We are interested in how to predict Yi when knowing Xi, where *i = t, ... L*. At the first glance, this seems a standard regression problem, but we know regression models overlook those temporal structures among Xi and those among Yi . Then it might be not a surprise that I will consider LSTM for the problem. Before we discuss the details, let’s look at a real-world dataset and a problem first.

This is a dataset introduced by Dr. Steve Scott (from Google) in his R package *bsts*. The target variable of interest, Yi, is weekly initial claims for unemployment insurance in the US, the exogenous variables, Xi, are the top 10 correlates generated by *Google Correlate*, also in the weekly granularity. The following plot shows the time series of weekly initial claims from 01/04/2004 to 09/30/2012, and the top correlates are, for example, the search queries of “Michigan Unemployment”, “Idaho Unemployment”, and so on over the same period.



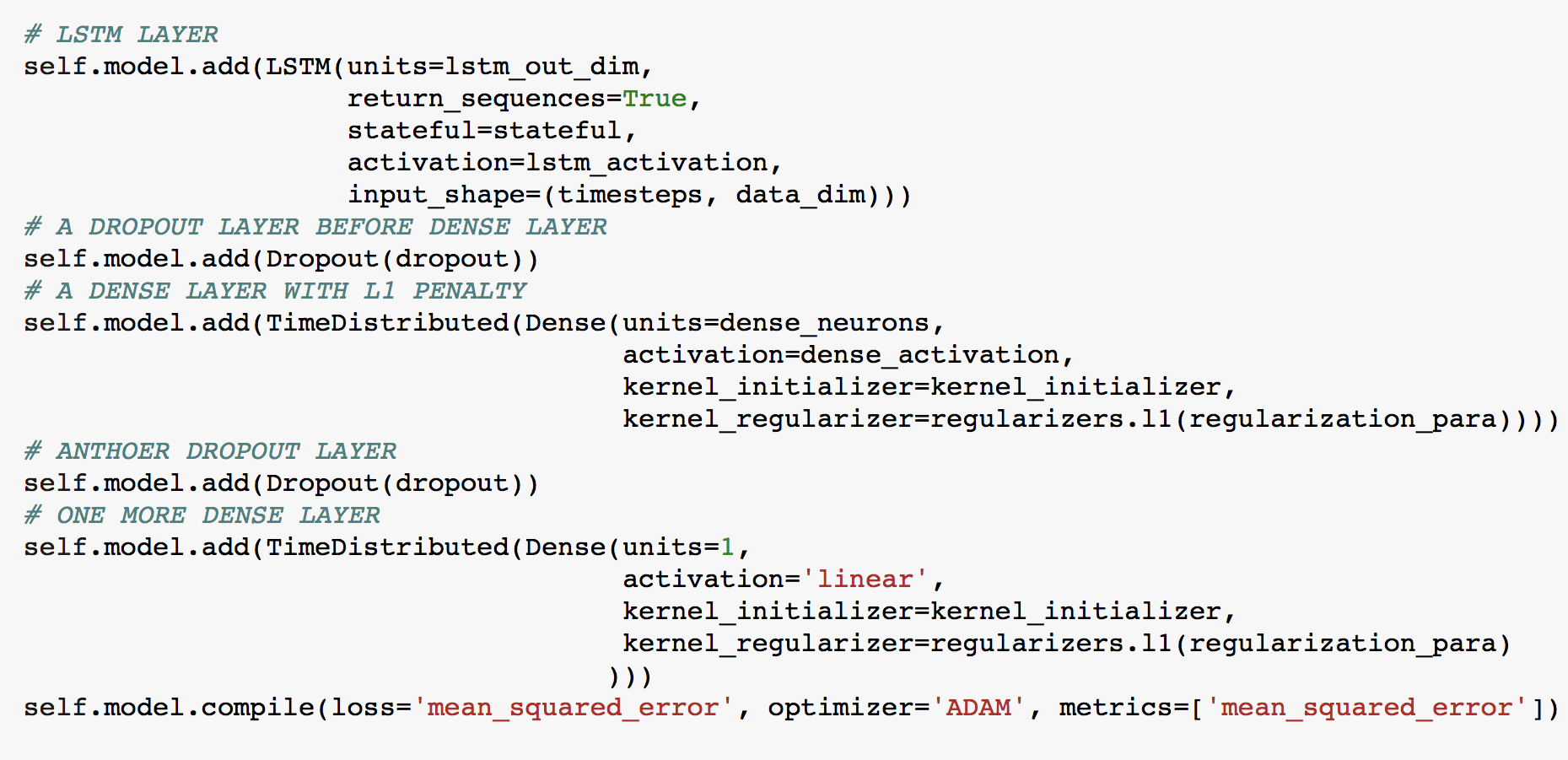
Because of the lag in collecting the data, even after a period of length *L*, we still can’t get Yt, ……, Yt+L, but we can obtain Xt, ……, Xt+L relatively quickly. Then we are interested in predict those Yi by Xi. Surely, there are many different means we can use, for example, regression models, ARIMA with regressors, State space model with regressors. Would LSTM be of help here?

Recall that an LSTM memory cell takes the current input and previous output to produce a new output. If this new output will be further used to predict the current target of interest, then this forms a many-to-many LSTM. The following plot shows the architecture I use for the insurance claim problem. The input of each LSTM memory cell, Xt , is the vector of top 10 Google Correlates at that time. ­ The output from the memory cell is then further fed into multiple dense layers, and then predict the current claims Yt



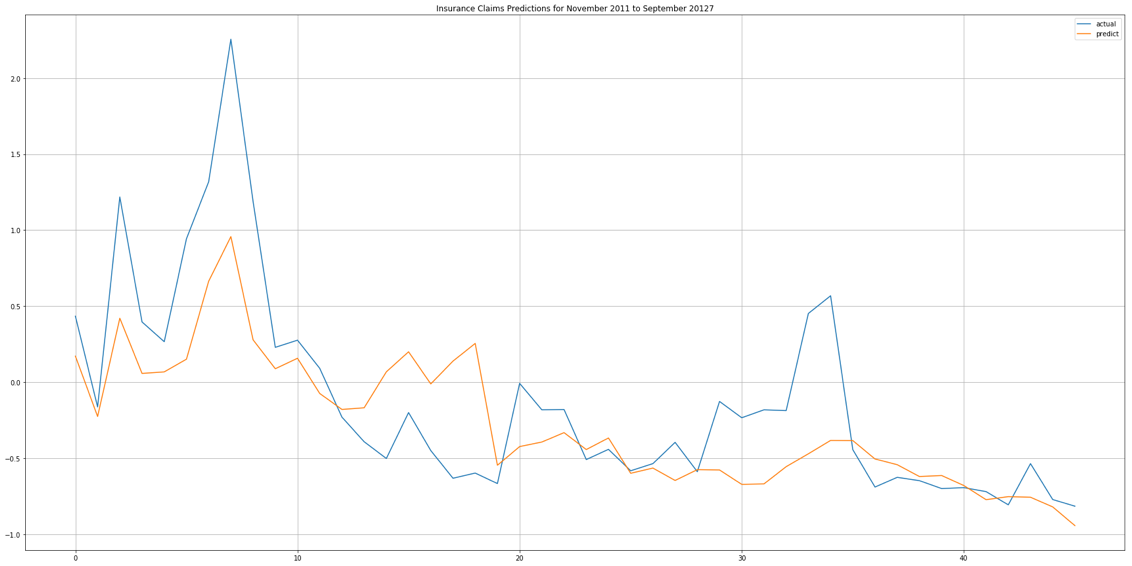
I hope this LSTM architecture will perform better than some of the models that only make the “independent” predictions like regression models. The reason is that the model not only uses the current explanatory variables but also consumes the past observations, and the network will figure out how much to remember from the history and how much to forget. We also hope this model is better than some of the linear models for example ARIMA, as it works in a non-linear fashion and can captures more complex dependence structures.

To make it a bit more concrete, the following code snippets, written in python and using Keras, show the model structure in details. “*self.model*” here is a standard Keras Sequential model. The first layer is a LSTM layer, then followed by two dense layers and form a pyramid shape. Because this is a very tiny dataset, to avoid overfitting and numerical issues, Dropout and weights penalties have been used.



Thanks for bearing with me. So far we have mainly looked at conceptual descriptions. Now let’s look at some example to demonstrate the idea. The claim dataset has 456 data points in total. Suppose we are interested in predict the last 46 weekly initial claims (about 11 months). I use the first 400 data points to train the model, and then make prediction on the last 11 months. There are many parameters in LSTM that we need to take care of, but here I might only mention the look-back length (I will have another post about visualization for deep learning models, and those visualization would provide some guidance on parameter tuning). I choose the look-back length as 7, pretty randomly. Once selecting the window length, I use a sliding window, and slice the training dataset to small pieces. Each of those pieces will be one data point for training the LSTM model. I slice the testing data, namely the last 11 months, in the same way. By a simple math, we can show for each of those predicted weeks there will be 7 predictions at most, then I take the average of those predictions as the final result.

The following plot shows the prediction results, where the blue line is the actual observations, and the orange line is the prediction. I might argue that the prediction seems to capture the overall trend and some of those bumps reasonably.



In summary, this article discusses a means to use LSTM to do time series forecasting for some special use case when we only have future exogenous variables available. This scenario can actually be found in many real-world problems. Hopefully the article has convinced you that Deep Learning could be an alternative to the “traditional” time series analysis techniques. I have put the source codes up to github, that include data preparation, model building, and prediction. Note that those are some quick codes, I will do some refactoring later to make codes more concise and efficient. Also note that the insurance claim data is not there. It is available in the R package of *bsts*. I know it will be a bit awkward to download a dataset from R and use it in Python. Finally I hope you will find this article be of any use.