UDACITY

Machine Learning Engineer Nanodegree

Capstone Project

Predicting U.S. Business Cycle Recessions with rNN

Author:
Chen Po-Chen

November 15, 2016

Contents

1 Definition

1.1 Project Overview

This capstone project attempt to use several type of neural network model to predict when recession will occur in business cycle in U.S.. Using a monthly time series data set covering from `1963:M1 to 2016:M9`, we investigate the impacts of some economic variables and financial variables on business cycle. We are going to find a neural network model which can predict recessions more precisely compared to regression, classification or time series methods, and to find out which type of neural network perform the best.

Business cycle turning points, or recession period, has always been deeply concerned by economists, government agencies and even by the general public. Government as well as business decisions rely on the assessment of the current business cycle. Provided the prediction of business cycle or recession, companies, governments, and investment organizations can calibrate their strategies, annual budget planning of next year or sales prediction...etc. This may directly help us to predict both on supply and demand side of how individual or companies will(should) act in the future which is critical in modern society.

It is hard but important problem to classify in which economic phase we are in. The `U.S. National Bureau of Economic Research (NBER)` [4] has established a standard operating procedure and regularly published the turning point date. In addition, in academia, economists make their effort in how to identify the business cycle accurately by economic or financial factors. These factors, or indicators, are anything that can be used to predict future financial or economic trends and can be classified to two types: `leading` and `lagging` indicators. In this project we will focus on the leading indicators. These types of indicators signal future events. In the world of finance, leading indicators work the same way as how we might know when we see the amber traffic light indicates the coming of the red light but are less accurate than the street light.

The essence of real-time turning point identification is a problem of statistical classification. Given a set of observations on economic indicators, we wish to determine which of two "classes" these observations came from, where the classes are expansion and recession. Much of the literature interested in identifying business cycle turning points has focused on parametric statistical models to link the observed data to the classes. For example, Chauvet (1998) proposes a dynamic factor model with Markov-switching (DFMS) to identify expansion and recession phases from a group of coincident indicators, and Chauvet and Hamilton (2006) and Chauvet and Piger

(2008) evaluate the performance of variants of this DFMS model to identify NBER turning points in real time. Fossati (2016) alternatively evaluates the real-time performance of a dynamic probit specification linking NBER expansion and recession phase indicators to observed data. If the true data generating process (DGP) linking the observed data to the classes is known, then such parametric models allow for the construction of an optimal Bayesian classifier of a period's data as belonging to either the expansion or recession class. [2] [5] [6]

1.2 Problem Statement

In this paper, we use a recurrent neural network to classify economic indicators as arising from either expansion or recession regimes in real time. Recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs. This makes them applicable to tasks such as unsegmented connected handwriting recognition or speech recognition. rNN takes both historical data and its classification as an input, which is then used to train the algorithm.

In addition, we will apply a network called long short-term memory (LSTM). Numerous researchers now use a deep learning RNN called the long short-term memory (LSTM) network. It is a deep learning system that unlike traditional RNNs doesn't have the vanishing gradient problem. LSTM is normally augmented by recurrent gates called forget gates. LSTM RNNs prevent backpropagated errors from vanishing or exploding. Instead errors can flow backwards through unlimited numbers of virtual layers in LSTM RNNs unfolded in space. That is, LSTM can learn "Very Deep Learning" tasks that require memories of events that happened thousands or even millions of discrete time steps ago. LSTM works even when there are long delays, and it can handle signals that have a mix of low and high frequency components and thus become very popular in context-sensitive language, machine translation, language modeling, and even time series model.

For this historical classification we use the available NBER chronology. Based on this training, new data that has not yet been classified by the NBER is then labeled as arising from the expansion or recession regime, which provides real-time tracking of new business cycle turning points. Our focus is on the ability of the algorithm to produce timely and accurate identification of new NBER business cycle turning points.

On the other hand, some researchers have used only one indicator such as national

aggregates of GDP or industrial production to define the business cycle. Such approaches, however, might undermine the multidimensional characteristics of business cycles, analysis of which requires broadly defined metrics that provide estimates for an underlying unobserved business cycle variable. In order to develop broadly defined metrics we use the coincident and leading economic indexes published by `NBER` and `Federal Reserve Bank of St.Louis (FRED)` [1].

Since the `industrial production index(IPI)` and the `S&P500(sp500)` are good reference of business cycle indicators, we consider these two factors to be our main explanatory variables which can predict the recession by the trend of industrial production index and the stock market return. The `Inflation rate(InRate)` is derived from `CPI` which helps the model to capture the influence of power of purchasing. `M1B` is the money supply or money stock which include the total of all physical currency including coinage plus the amount of demand deposits, travelers checks and other checkable deposits, in other words we only consider the 'currencies' with high liquidity. The last variable is `New Private Housing Units Authorized by Building Permits(Permit)` which is also a good and leading indicator that depict builders will apply new permits for their new project whenever it is in the blooming period of business cycle.

The economic and financial variables which are included in the prediction model are listed in Table 1. We concentrate on the predictive power of these variables for business cycle and the recessions. The advantage of those variables is that those are available on a continuous basis without revisions. Because of informational lags and revisions of important macroeconomic variables, the current state of the economy is always uncertain to some extent. Thus we are interested in predicting the probabilities of `recessions(Rec)` in a business cycle for month t using the information up to the end of the previous month $t - n$ where we will use n equals to 3(a quarter lag), 6(half year lag), 12(one year lag). In other words, the nowcasts are constructed at the beginning of month t.

Table 1: Variables

Variable	Period	Frequency	Data Type	Denote	
`Rec`	1963:M1-2016:M9	Month	Binary	y	Recession
`IPI`	1963:M1-2016:M9	Month	Continuous	X_1	Industrial Production Index
`sp500`	1963:M1-2016:M9	Month	Continuous	X_2	S&P500 Index
`Permit`	1963:M1-2016:M9	Month	Continuous	X_3	New Private Housing Units Authorized by Building Permits
`M1B`	1963:M1-2016:M9	Month	Continuous	X_4	M1B
`InRate`	1963:M1-2016:M9	Month	Continuous	X_5	Inflation Rate

Data source:
[1] National Bureau of Economic Research
[2] Federal Reserve Bank of St. Louis
[3] y denote dependent variable and X denote independent variables

1.3 Metrics

Since there's quite difference between using deep learning and traditional econometric models, we will consider neither AIC nor BIC to evaluate our models with lag terms. In order to make our evaluation rigorous, we will choose R^2 to evaluate models on training set and $F1$ score to evaluate models on testing set. Though R^2 and $F1$ score are thought to be simple bench marks compared with AIC and BIC in the scenario of predicting time series, R^2 and $F1$ score are easier to interpret while AIC and BIC are not available in `scikit-learn` [3] or `keras`.[1] As a result, we will use $F1$ score as first metric then the R^2 in order to measure not only the predictive power but how the classifiers depict variance of data. And also that's why we use mean square error in our loss function instead of other choices.

2 Analysis

In this section, we will discuss the data which are used to construct our models, the meaning of exploratory visualization in subsection 2.2 will be explained and cited in the context of subsection 2.1.

[1]There's do have AIC, BIC features in Lasso model selection, but it's doubtable to apply to deep learning

2.1 Data Exploration

Table 2: Descriptive Statistics

stats	rec	permit	m1b	ipi	cpi	sp500
count	681	681	681	681	681	681
mean	0.14	1357.62	894.79	66.01	121.31	584.85
std	0.34	395.74	746.79	25.79	70.05	591.74
min	0	513	139.60	22.972800	29.37	53.39
25%	0	1078	266.70	45.13	47.80	98.44
50%	0	1340	771.70	63.05	117.50	284.20
75%	0	1655	1196.40	93.81	180	1111.92
max	1	2419	3317.90	106.68	241.01	2173.60

In Table 2 shows the descriptive statistics of Recession(`Rec`), Industrial Production Index(`IPI`), S&P500(`sp500`), New Private Housing Units Authorized by Building Permits(`Permit`), Inflation Rate(`lnRate`) from 1963:M1 to 2016:M9. We got 681 data points with each variables before `natural logarithmic first order difference transformation(NLD)`. The various of range of these variables could cause inefficient to our algorithms, so we will use the `max-min scalar` in `scikit-learn` to rescale our data to [0, 1] interval.

Figure 1 shows the trend of those five observed variables. At the first look of Figure 1, we might easily get a idea that four of five variables are trending and `permit` is volatile. However, while we are using time series variables to predict something, the concept of 'change' is much more powerful to depict what happened then the 'static status'. Logarithms are often a much more useful way to look at economic data. For example, the `sp500` in Figure 1. Plotted on this scale, one can see nothing in the first century, whereas the most recent decade appears insanely volatile. On the other hand, if we plot these same data on a log difference scale, a vertical move of 0.01 corresponds to a 1% change at any point in the figure. Plotted this way, it's clear that, in percentage terms, the recent volatility of stock prices is actually modest relative to what happened in 1988 and 2008. This is the advantage of `NLD` result from the `natural logarithm` part.

Furthermore, in Figure 3 and Figure 4, we may see the correlation coefficient between each variables before and after `NLD`. In particular, we may find that `M1B` has highly correlated with `IPI`, `CPI` and `sp500` and are respectively 0.878, 0.923 and 0.927. This

fact may cause a problem of endogeneity. However, after `NLD` transformation, not only the problem of endogeneity has been eased but the importance of correlations of `Rec` with each observed variables are standing out. On the other hand, `first order difference` also helps a lot with dealing with unit root problems in time series data. A unit root is a feature of some stochastic processes that can cause problems in statistical inference involving time series models, and the first order difference of the process will eliminate the stochastic trend of time series and transform to be stationary where only the deterministic trend will be remained.

Table 3: Correlation coefficient matrix

	rec	permit	m1b	ipi	lnRate	sp500
rec	1.000000	-0.330341	-0.132566	-0.100110	-0.085488	-0.125647
permit	-0.330341	1.000000	-0.211522	0.016049	-0.119469	-0.068916
m1b	-0.132566	-0.211522	1.000000	0.877502	0.923262	0.927279
ipi	-0.100110	0.016049	0.877502	1.000000	0.974809	0.925309
cpi	-0.085488	-0.119469	0.923262	0.974809	1.000000	0.910677
sp500	-0.125647	-0.068916	0.927279	0.925309	0.910677	1.000000

Table 4: Correlation coefficient matrix after NLD

	rec	permit	m1b	ipi	lnRate	sp500
rec	1.000000	-0.075076	0.047707	-0.501322	0.126181	-0.085888
permit	-0.075076	1.000000	0.083352	0.141482	-0.041104	0.081412
m1b	0.047707	0.083352	1.000000	-0.106642	-0.062900	-0.040371
ipi	-0.501322	0.141482	-0.106642	1.000000	-0.028083	0.004545
cpi	0.126181	-0.041104	-0.062900	-0.028083	1.000000	-0.093097
sp500	-0.085888	0.081412	-0.040371	0.004545	-0.093097	1.000000

2.2 Exploratory Visualization

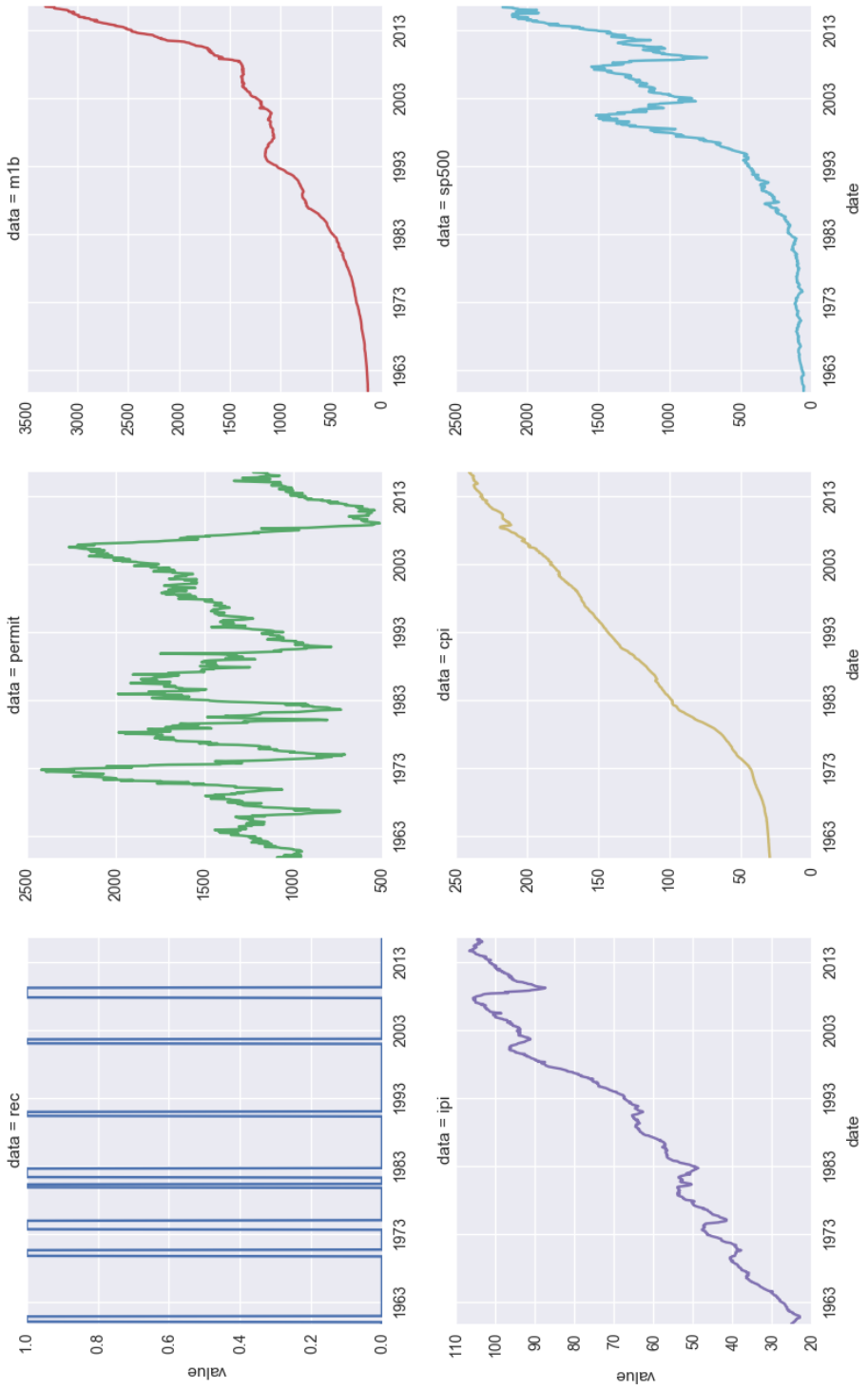


Figure 1: Time Series Data

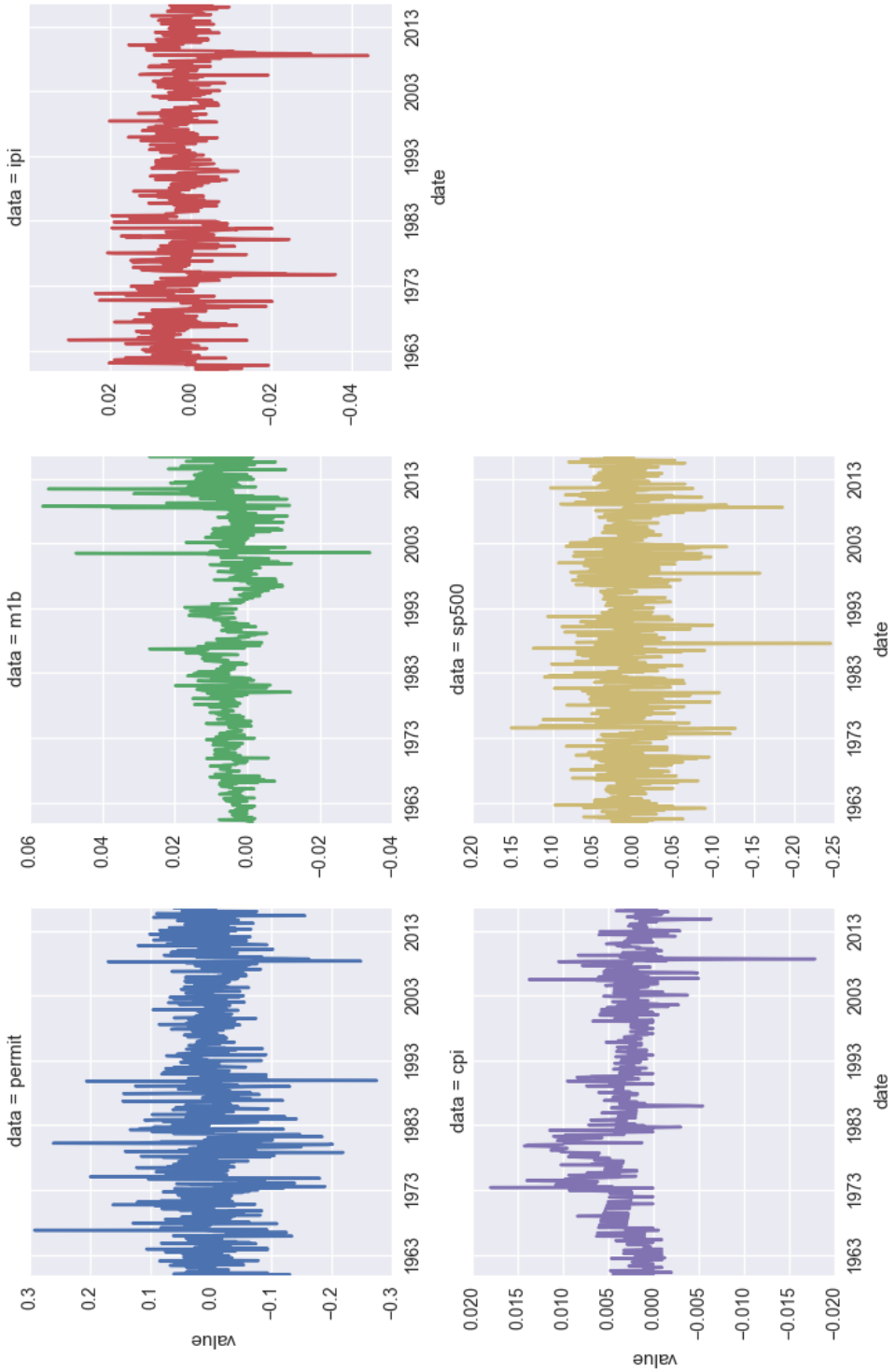


Figure 2: Time Series Data after NLD

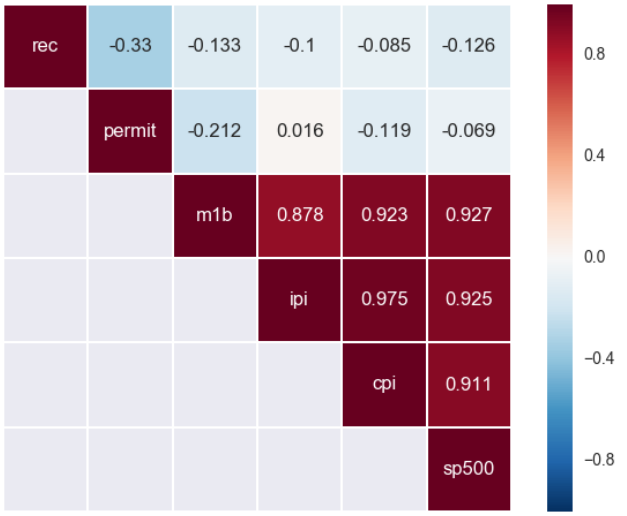


Figure 3: Correlation coefficient matrix of observation variables

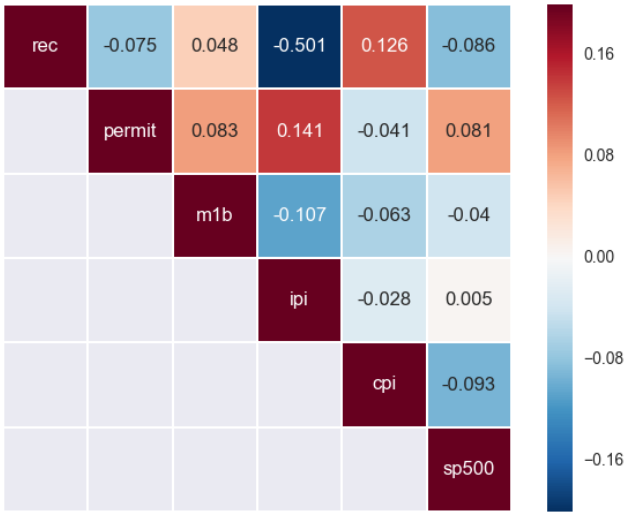


Figure 4: Correlation coefficient matrix of observation variables after NLD

2.3 Algorithms and Techniques

Model 1 (Multilayer Perceptron Regression)

A `multilayer perceptron (MLP)` is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs as Figure 5 presenting. It consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. When Back propagation is used in very deep neural networks and in unrolled recurrent neural networks, the gradients that are calculated in order to update the weights can become unstable. They can become very large numbers called exploding gradients or very small numbers called the vanishing gradient problem. These large numbers in turn are used to update the weights in the network, making training unstable and the network unreliable. We'll see the limitations of `MLP` that are addressed by recurrent neural networks in this Many-to-one problems due to these issues.

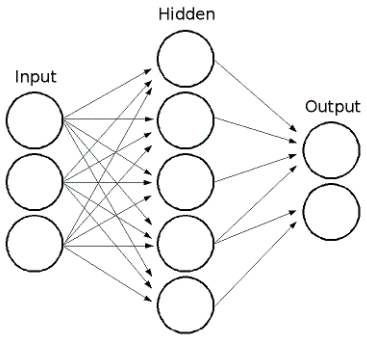


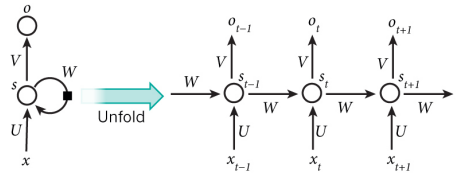
Figure 5: Circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.

Model 2 (LSTM Network for rNNs)

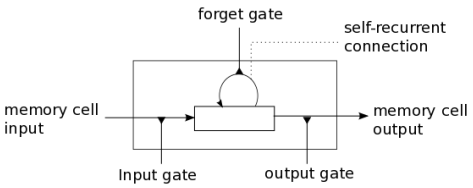
Before we start to introduce how to deal with the issue of gradient vanishing, let's walk through what is recurrent neural network first. Recurrent neural network is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network which allows it to exhibit dynamic temporal behavior. rNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. In other words, rNNs have a "memory" which captures information about what has been calculated so far.

To deal with the issues we mentioned above in recurrent neural network architectures, we can alleviate the problems by using a new type of architecture called the `Long Short-Term Memory Networks` that allows deep recurrent networks to

be trained. The `Long Short-Term Memory` or `LSTM` network is a recurrent neural network that is trained using Back propagation Through Time and overcomes the vanishing gradient problem. As such it can be used to create large (stacked) recurrent networks, that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results. Instead of neurons, `LSTM` networks have memory blocks that are connected into layers.



(a) A recurrent neural network and the unfolding in time of the computation involved in its forward computation.



(b) A neuron of rNN

Figure 6: Concept of rNNs and its single neuron

Model 3 (Multi-Dimensional LSTM rNNs)

Not only the gradient problem that we need to solve but also how to maximize the explanatory power of variables we use to train our `LSTM` model. We will use parallel sequences of LSTMs(`pLSTM`) to first deal with these 5 variables respectively and then merged into one sequence of neural network model. The concept is as figure 7 [7].

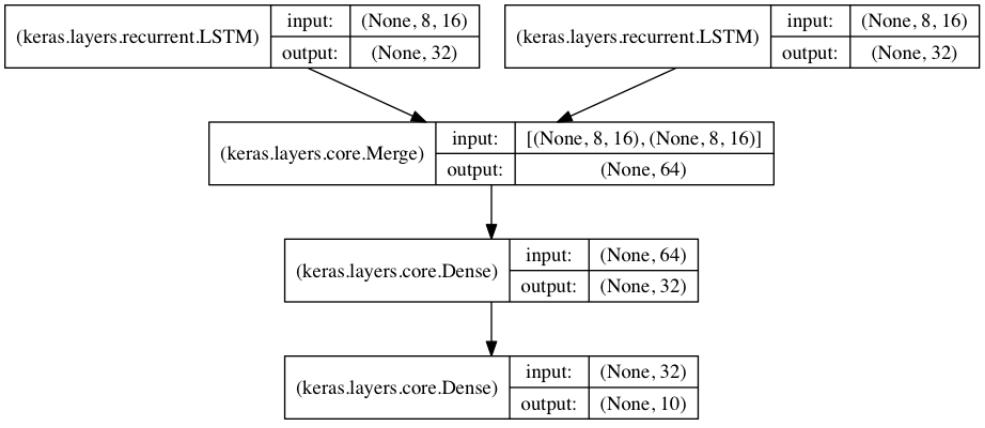


Figure 7: Concept of Sequences LSTM

Model 4 (Multi-Dimensional Stateful LSTM rNNs)

The LSTM network has memory which is capable of remembering across long sequences. Normally, the state within the network is reset after each training batch when fitting the model. We can gain finer control over when the internal state of the LSTM network is cleared in Keras by making the LSTM layer stateful which is the mainly difference between Model 3, `psLSTM`. This means that it can build state over the entire training sequence and even maintain that state if needed to make predictions.

The purpose we construct Model 1(`MLP`) and Model 2(`LSTM`) in this project is to verify the power of `LSTM` and how it perform after solving the problem of `gradient vanishing`. Secondly, comparing the performance of Model 2 to Model 3(`pLSTM`) and Model 4(`psLSTM`), we might find out whether the performance of pooling each variables after we verified their relation to `Rec` respectively is better than the performance of pooling all variables together at first hand or not.

2.4 Benchmark

Since there are no AIC and BIC in the scenario of deep learning as mentioned in 1.3, and as such it is not possible to set an experimentally justified benchmark for the data set. As such, it is necessary to use other means by which to set a benchmark to determine whether or not the algorithms used for this problem are successful.

The University of Nebraska Medical Center has an interesting article discussing the use of the ROC curve for interpreting the usefulness of a diagnostic test in the field of medicine. In their estimation, an effective guide for interpreting an roc_auc_score in order to establish the success of a given test at correctly differentiating between disease and non-disease is the "traditional academic point system". $F1$ score is applicable for any particular point of the ROC curve. This point may represent for example a particular threshold value in a binary classifier and thus corresponds to a particular value of precision and recall, and thus I will set the benchmark as follow:

Table 5: Benchmark

$.90 - 1 =$ excellent (A)
$.80 - .90 =$ good (B)
$.70 - .80 =$ fair (C)
$.60 - .70 =$ poor (D)
$0 - .60 =$ fail (F)

Of course, getting a diagnosis wrong in a medical classification procedure is going to be far more problematic than getting a diagnosis wrong in classifying an economic event. However, it seems reasonable to go by a similar point system in determining the success of an algorithm. With this in mind, a roc_auc_score of .85 or higher will be considered an excellent result for the algorithm, while a score of .80 to .85 will be considered a good result.

3 Methodology

3.1 Data Preprocessing

There was no missing data included in the given data set, so no preprocessing work was required to fill in missing values. However, it will produce 3, 6, 12 missing values each columns after we did the first order difference logarithmic transformation, and so we will simply drop these NA rows.

Since the scale of values differs amongst the features, a step was included to normalize the data – that is, each column was scaled so that the resulting column to [0, 1] interval.

3.2 Implementation

The steps needed to be implemented to solve the problem are as follow: load and prepare the data, perform the statistical analysis, preprocess the data, train the classifiers, and output the results.

The first step in the implementation of the learning algorithms was to split the data set into a training set and a testing set using the sklearn function `train_test_split` from the package `sklearn.cross_validation` 80% of the data was reserved for training the classifier, while **20**% was set aside for testing the classifier once it has been trained. This split was chosen due to the abundance of data in the data set – if the data set had been smaller, reserving 20% for testing might not have been feasible. Note that this split happens after the preprocessing of the data has been completed, and also since it's a time series problem we are not going to shuffle the data set so as to keep the orders.

Once the data set had been split up into training and testing sets, the classifiers could be built and trained in the classifier. We will choose `adam` or `sgd` as our gradient algorithms and simply set batch equals to 1 since we will 'control the batch' by using

different lag periods(3, 6 ,12) to control the learning rate. The primary complication that occurred during the implementation phase was related to the setting of layers on deep learning models. It is very difficult to perform a very thorough search of the possible numbers, well-performed layers and types of activation functions in a reasonable amount of time to trial and error.

The learning rate we're using is the keras' default setting which is 0.001 in `adam` and 0.01 in `sgd`, and also the decay rate and epsilon are using default setting. The reason we did not tune the parameters is because it seldom stuck in the same gradient and also provide a reasonable speed in gradient descend. The way we examine whether we stuck in local optimal is to change the random seed. However, this definitely would be the trivial part that we can improve.

The final layers designs are as Figure 8, Figure 9, Figure 10 and Figure 11 in 3.4 `Appendix:Model Design`.

3.3 Refinement

The Model 4 `psLSTM` was selected for further refinement of Model 3 `pLSTM`. The refinement step will be to simply dig down deeper and add the `stateful=True` into the parameter space in order to utilize the memory feature of LSTM to produce the best possible neural network.

3.4 Appendix: Model Design

dense_1 (Dense)	input	Dim(x)	activation	relu
	output	(None, 32)		
↓				
dense_2 (Dense)	input	32	activation	relu
	output	(None, 64)		
↓				
dropout_1 (Dropout)	input	64	activation	-
	output	(None, 64)		
↓				
dense_3 (Dense)	input	64	activation	relu
	output	(None, 8)		
↓				
dense_4 (Dense)	input	8	activation	sigmoid
	output	(None, 1)		

Figure 8: MLP

lstm_1 (LSTM)	input	Dim(x)	activation	tanh
	output	(None, None, 8)		
↓				
lstm_2 (LSTM)	input	32	activation	tanh
	output	(None, 4)		
↓				
dropout_1 (Dropout)	input	64	activation	-
	output	(None, 4)		
↓				
dense_1 (Dense)	input	64	activation	relu
	output	(None, 4)		
↓				
dense_2 (Dense)	input	8	activation	sigmoid
	output	(None, 1)		

Figure 9: LSTM

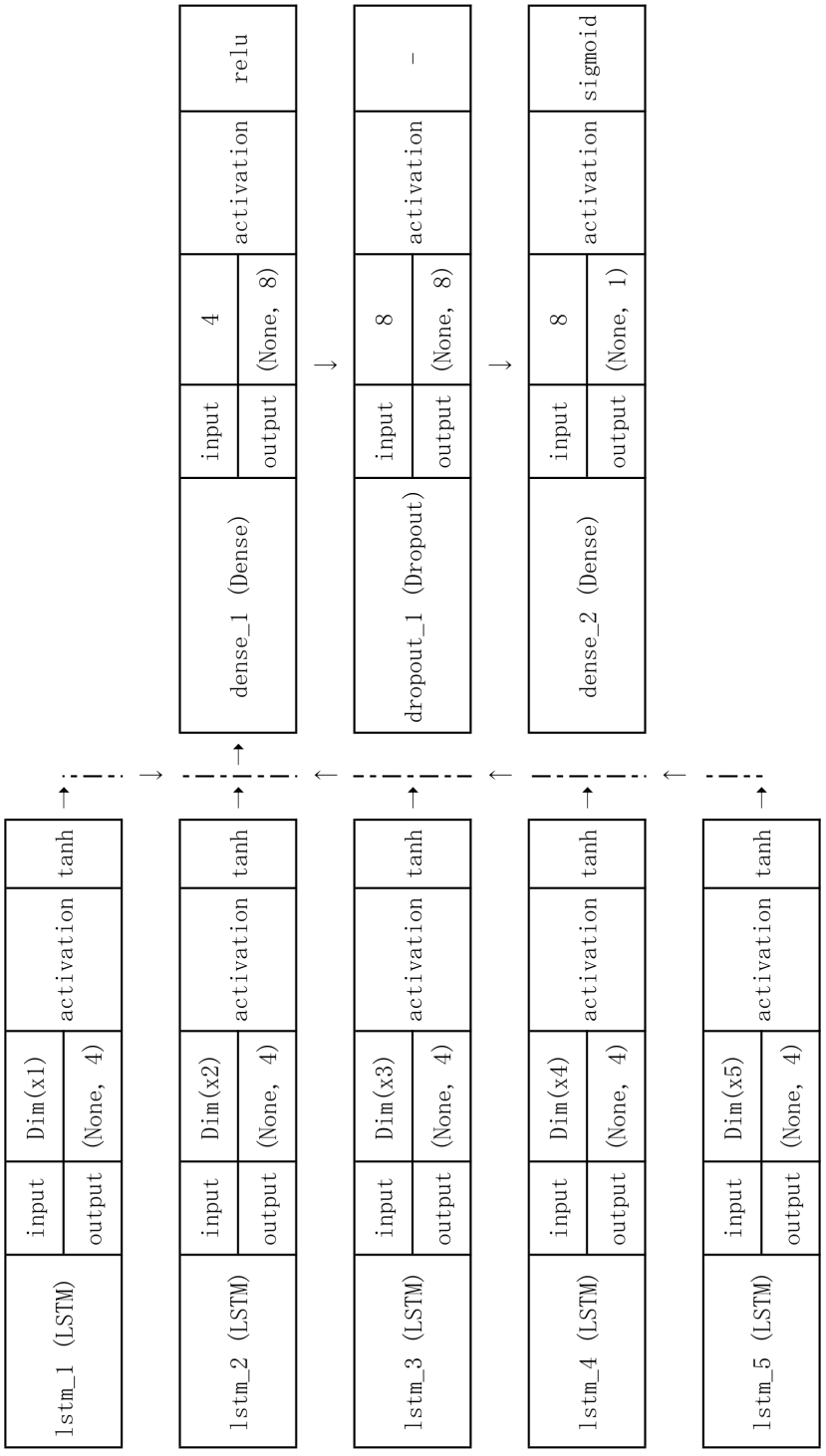


Figure 10: Model design: pLSTM

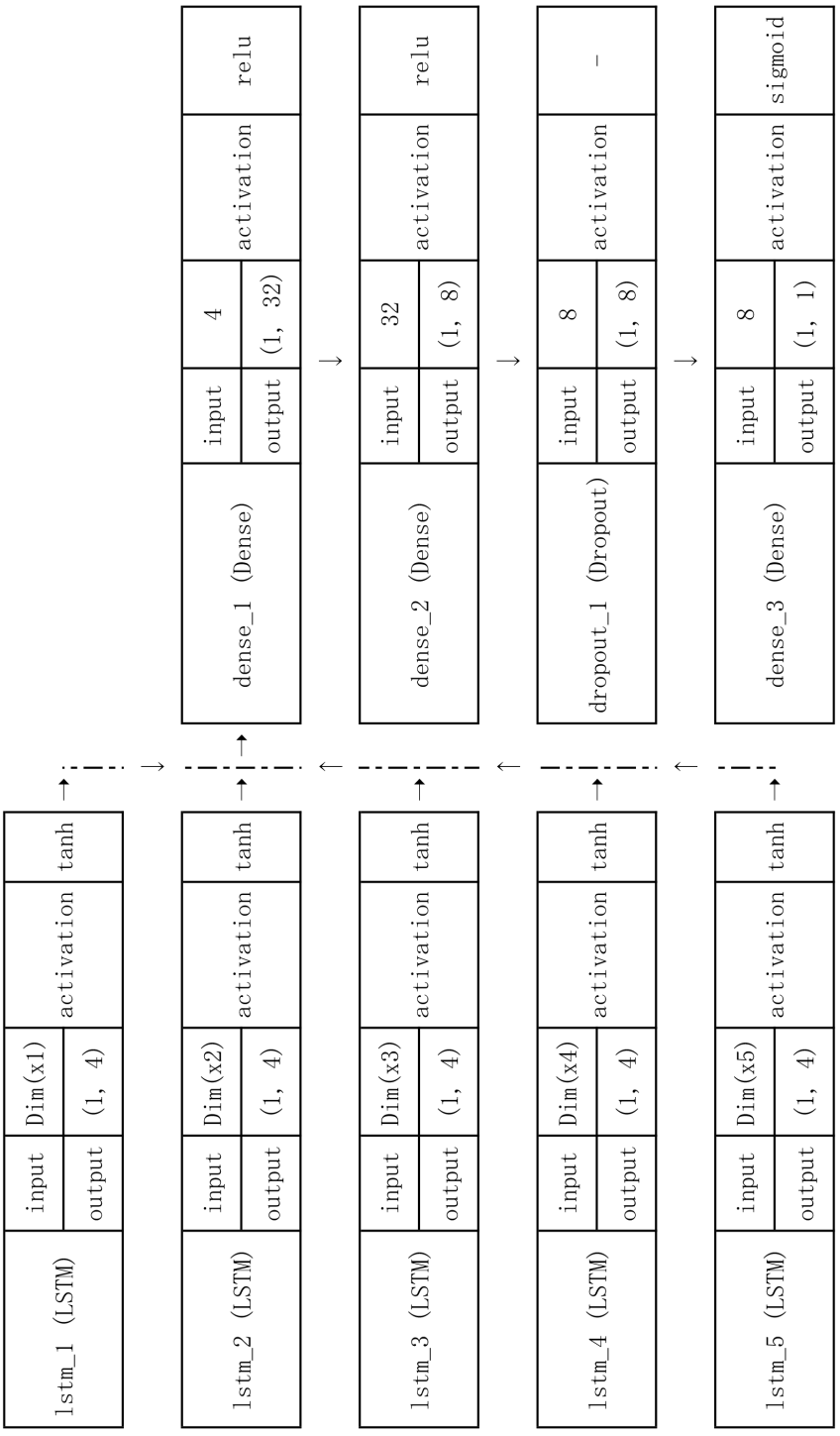


Figure 11: Model design: psLSTM

4 Results

4.1 Model Evaluation and Validation

Training and testing times for each of the neural networks attempted is given in Table 6. Clearly, the `psLSTM` is by far the slowest of the neural networks in terms of both training and testing times. The training time for the `MLP` is the smallest with no surprise, while that of the `pLSTM` almost equals to the `psLSTM`.

Table 6: Training and testing times for the various models

	`MLP`	`LSTM`	`pLSTM`	`psLSTM`
Training Time	153s	1503s	4501s	4802s
Testing Time	1.01s	3.10s	5.11s	5.12s

† With GPU acceleration

Table 7 shows the results of the neural networks when trained and tested on the train data and reserved test data repectively (a visualization of the results can be seen in the in-sample prediction of time series given in Figure 12, 13, 14 and 15 in 5.1. It is evident that the `psLSTM` perform better than `pLSTM` in terms of R^2. In this and nearly every run that was undertaken, the `psLSTM` also slightly outperformed the `pLSTM` in terms of $F1$ score.

Table 7: Training and testing scores for the various models

Models	Parameters	Depth	Parallel	Stateful	R^2	$F1$ Score
`MLP`	lag=12, epoch=150	5	-	-	0.518	32.94%
`LSTM`	lag=12, epoch=150	5	F	F	0.652	79.75%
`pLSTM`	lag=12, epoch=150	4	T	F	0.901	93.49%
`psLSTM`	lag=3, epoch=1, loop=150	5	T	T	0.958	95.23%

[1] R^2 evaluate the probability outputs
[2] $F1$ Score evaluate the binary outputs
[3] batch=1

Comparing to the benchmark we set in Sec. 2.4 Table 5, we can see the final result in Table 8. And thus we might conclude that the final model chosen for the problem is the `psLSTM`. In particular to the questions we've asked in Sec.2.3, first of all, we may observe that the performance of `LSTM` is better than `MLP` in both R^2 and $F1$ score as expected. Also, it is obvious that with a feature of parallel sequences such as `pLSTM` and `psLSTM` is extremely outperformed the `LSTM`.

Table 8: Beat Benchmark

$.90 - 1 =$ excellent (A)	`pLSTM` , `psLSTM`
$.70 - .80 =$ fair (C)	`LSTM`
$0 - .60 =$ fail (F)	`MLP`

4.2 Justification

It is clear, from Table 7, that both `MPL` and `LSTM` model were not sufficient to solve this problem – it seems highly unlikely that even with further refinement this model would perform to the 'excellent' level set by the benchmark. The `pLSTM` and `psLSTM` model, however, could possibly be made 'excellent' with further refinement, although the extremely long training times for this algorithm (see Table 6) make further refinement extraordinarily difficult.

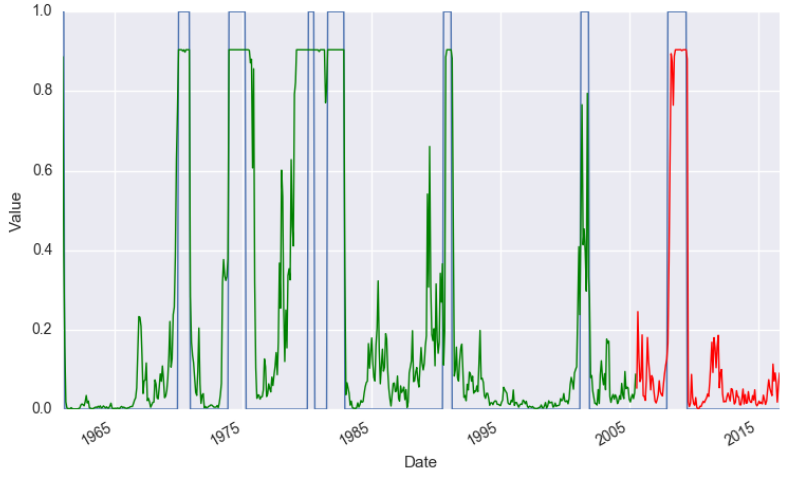
5 Conclusion

5.1 Free-Form Visualization: In-sample Prediction

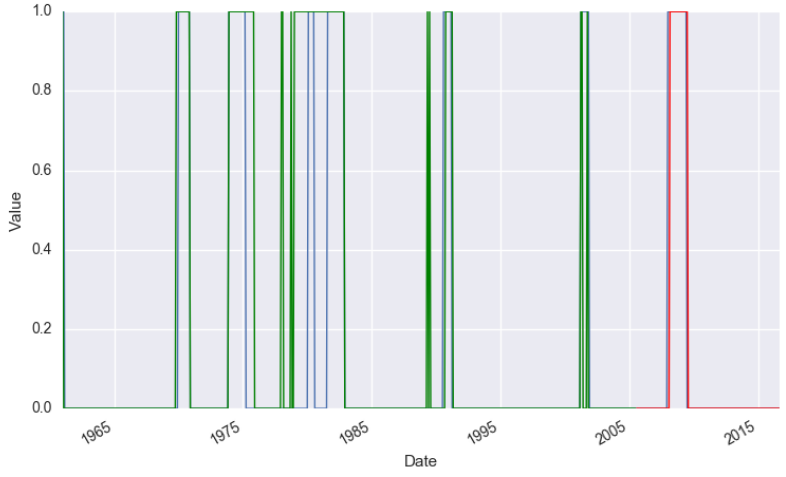
We've discussed the predictive power of four of out models in Section 4.1 with compared to the benchmarks we set up. In this section, we are going to see the figures which is depicted the time series plot of real and predictive value. An predictive time series was produced that compared to the actual value of recession in order to develop a visualization of our original problem.

Since the predictive outcomes are probabilities of recession will happen in each time t, we will simply transform our prediction to binary form so as to 'guess' whether it is recession or not. If the probability is larger than or equals to 0.5, we'll set `rec` equals to 1, otherwise `rec` is 0 which also means it's in expansion period, that is:

$$\texttt{Rec\_binary}_t = \begin{cases} 1, & \text{if the } \texttt{Rec}_t \geq 0.5 \\ 0, & \text{if the } \texttt{Rec}_t < 0.5 \end{cases}, \text{ where } \texttt{Rec} \text{ is in the probability form.}$$

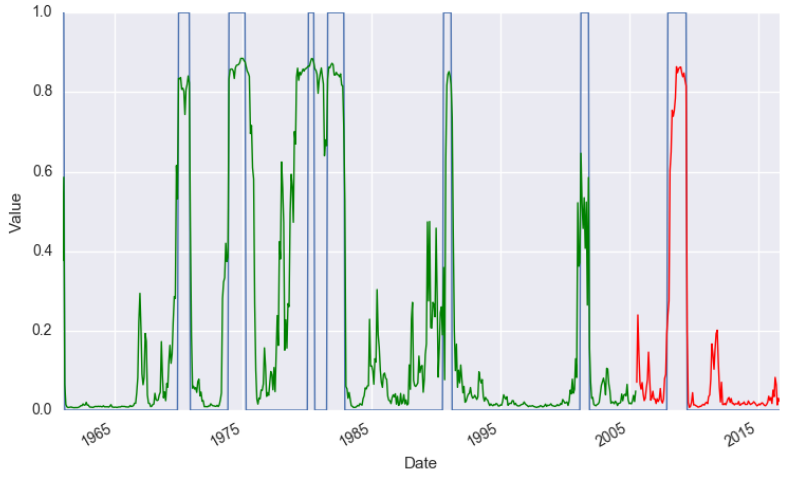


(a) Probability Output

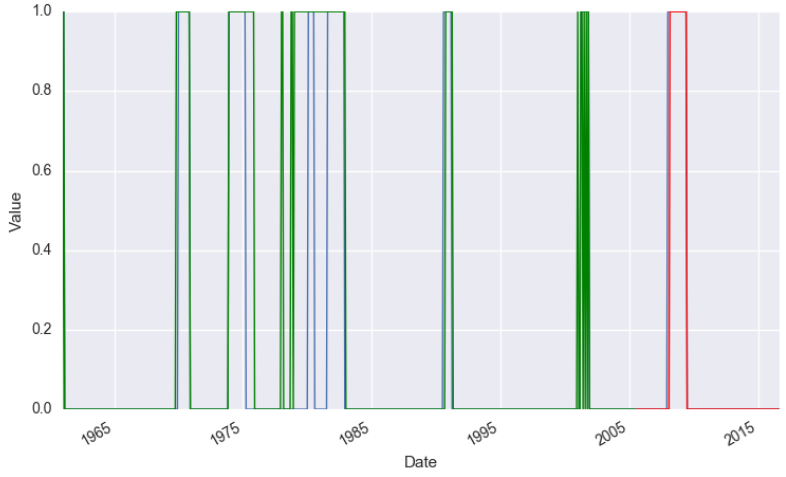


(b) Binary Output

Figure 12: Outputs of Multilayer Perceptron Regression

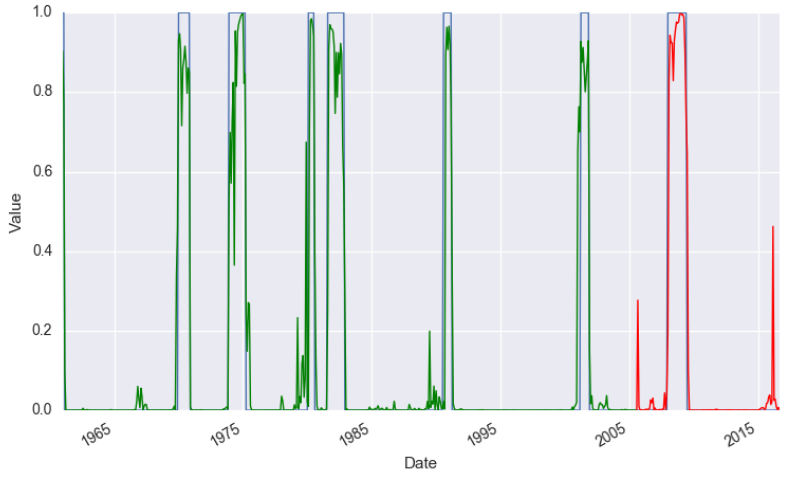


(a) Probability Output

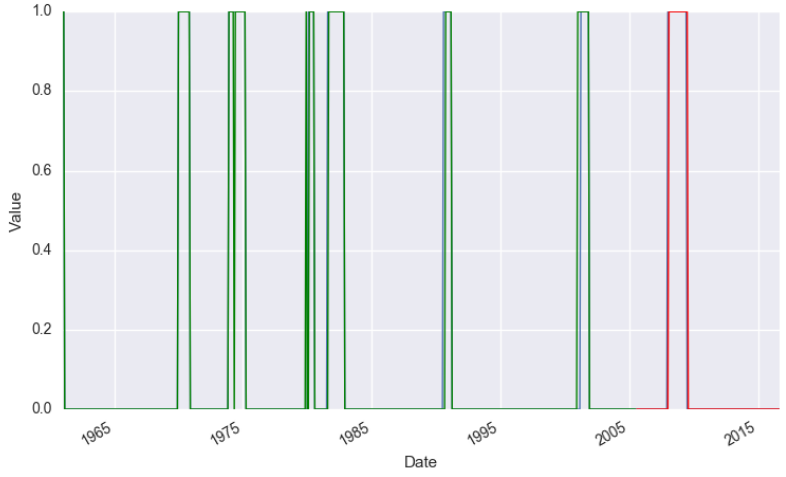


(b) Binary Output

Figure 13: Outputs of LSTM with Single Layer Model

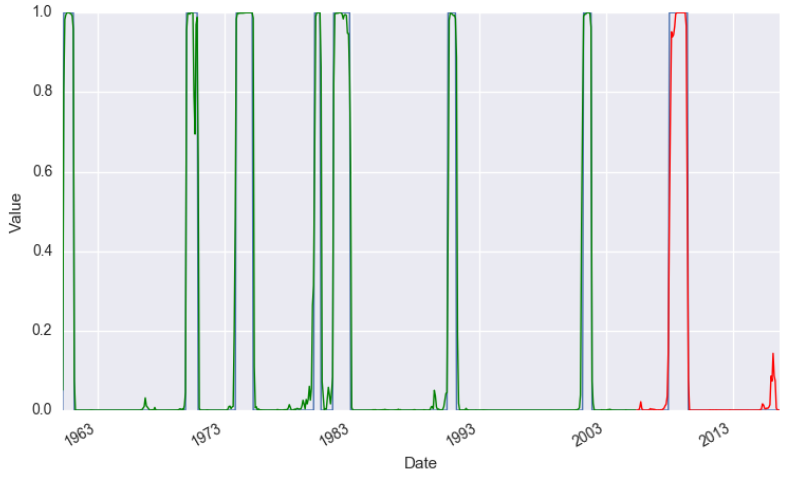


(a) Probability Output

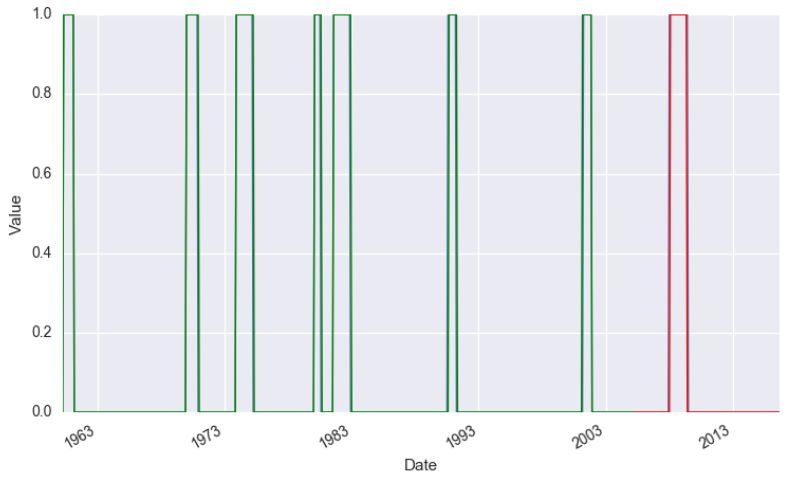


(b) Binary Output

Figure 14: Outputs of Multi-Dimensional LSTM Model



(a) Probability Output



(b) Binary Output

Figure 15: Outputs of Multi-Dimensional Stateful LSTM Model

5.2 Reflection

Machine learning algorithms are commonly used in many disciplines to classify data as belonging to one of a set of classes. We have proposed a particularly salient algorithm, known as recurrent Neural Network, for the purposes of classifying economic data into expansion and recession regimes. Of particular interest is the ability of the algorithm to accurately and quickly identify U.S. business cycle turning points

in real time.

The project began with a statistical analysis of the data set, from which it was determined that rescaling and normalization of the data was appropriate. Once the preprocessing was done, the data set was split into training and test sets, with 20% of the data set aside for testing the various models. Next, several neural networks were trained, using manually tuning. Once several neural networks were trained, the results of the neural networks on the test data set were plotted and compared using an time series curve with in-sample test. From the result of R^2 and $F1$ score table, it was determined that `psLSTM` was the best model to use for this problem, although the `pLSTM` also produced good results.

We evaluated the real-time performance of the rNN with LSTM algorithm for identifying business cycle turning points in the United States over the past 50 years and eight recessions. The `pLSTM` and `psLSTM` identified the timing of all eight recessions over this period accurately. In addition, looking across all recessions, the algorithm's speed for identifying peaks and troughs over our sample period is very competitive with the dynamic factor `Markov-switching model`, a commonly used technique for dating business cycles in real time.

The most difficult part of the project was determining activation functions, lag periods and metrics to evaluate, since it's a heavy time consuming work to try each combination given that deep learning is already a burden. In addition to the selection of the parameters to search, it was necessary to come up with how many layers we need, should it be deeper? or should it be wider? In some cases, this was easy – for example, in the `MLP` and `LSTM` models' layers design, since I just want to make a simple comparison of the outcome before and after dealing with `gradient vanishing`. But for others, `pLSTM` and `psLSTM` are the main models in this project which we would like to use them to predict the outcomes. Since there's no package with grid search for deep learning and manually tuning is exhaustive, adding even a few values increases the search time dramatically. So it was difficult to check all possible width and length of neural networks – a lot of trial and error was involved in the selection process.

5.3 Improvement

The essential way to improve this project is either my knowledge to deep learning or GPU in my PC need to be upgraded. A better domain knowledge of deep learning with better equipment reduce the time not only computation needed but elimination the models we even need not to try. On the other hand, we need to improve the metrics that we used to choose best model. In order to improve the model selection

strategy, we should implement AIC and BIC test and apply to the last layer, for instance, of neural networks to help to evaluate our models. And These are also the difficult part in this project.

Secondly, we would have to tune our optimizer like `adam` or `sgd` in order to make our process robust and might speed up the process as well. Furthermore, we would like to add more predictors such as real investment in equipment, real private consumption, unit labor cost, growth rate of wage and salary earners, long and short term interest rate, government deficit, and house affordable index. Some of these are really good leading indicators while some of them are not free to reach especially the house affordable index is the most powerful leading indicator in these factors that we didn't utilize but also the most expansive one.

Finally, one algorithm that was not attempted, but that might very well perform better than the ones tried in this project, is the rNN plus convolutional Neural Network. convolutional Neural Networks are very powerful algorithms and could be used to replace the middle and final layers in our original networks. It was felt that they were probably more complex than was required for a good solution to this problem. However, since this combination is more powerful, it is certainly feasible that the new benchmark achieved in this project using the `pLSTM` and `psLSTM` might be eclipsed by a Neural Network implementation.

References

[1] Federal reserved bank of st. louis economic data. https://fred.stlouisfed.org. Accessed: 2016-08-30.

[2] Has the us economy become more stable? a bayesian approach based on a markov-switching model of the business cycle. https://goo.gl/XWEIlb. Accessed: 2016-11-11.

[3] Lasso model selection: Cross-validation / aic / bic. https://goo.gl/51kEin. Accessed: 2016-10-30.

[4] National bureau of economic research (nber). http://www.nber.org. Accessed: 2016-08-30.

[5] A new approach to the economic analysis of nonstationary time series and the business cycle. https://goo.gl/SkQaea. Accessed: 2016-11-11.

[6] Predicting recessions with leading indicators by fred. https://goo.gl/wq2Gn4. Accessed: 2016-11-11.

[7] Two merged lstm encoders for classification over two parallel sequences. https://goo.gl/qKgBjg. Accessed: 2016-10-30.