Time Series Forecasting Using Neural Network-Based and Time Series Statistic Models

Jeffrey Yau
Chief Data Scientist, AllianceBernstein, L.P.
Lecturer, UC Berkeley Masters of Information Data Science

Learning Objectives

For data scientists and practitioners conducting time series forecasting

After this introductory lecture today, you will have learned

1. The characteristics of time series forecasting problem
2. How to set up a time-series forecasting problem
3. The basic intuition of two popular statistical time series models and LSTM network
4. How to implement these methods in Python
5. The advantages and potential drawbacks of these methods

Agenda

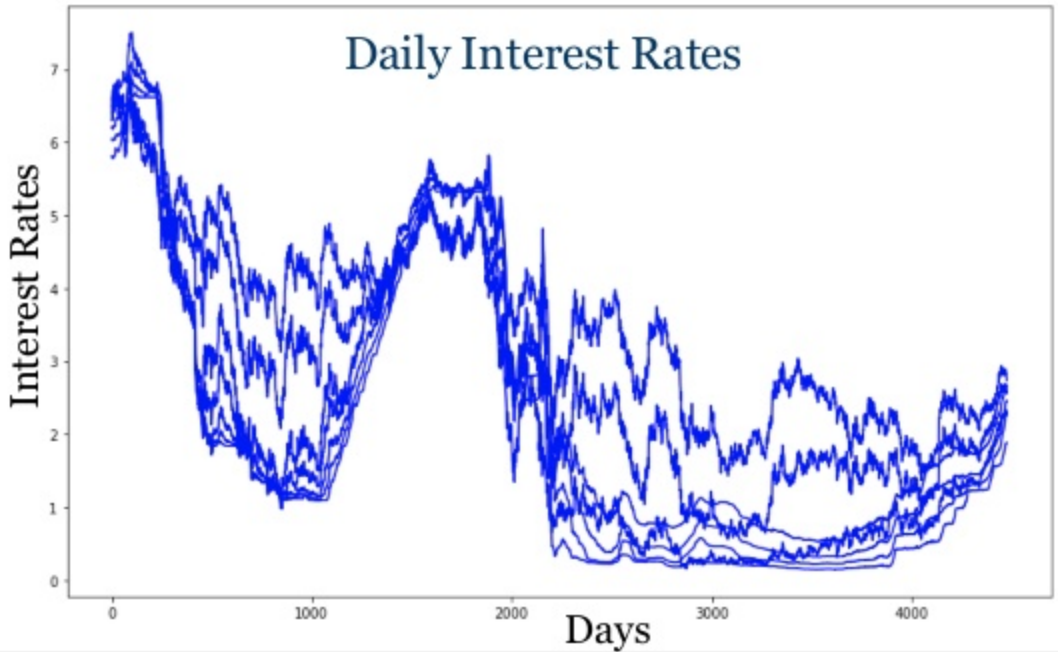
Section I: Time series forecasting problem formulation
Section II: Univariate & Multivariate time series forecasting
Section III: Selected approaches to this problem:

- ❖ Autoregressive Integrated Moving Average (ARIMA) Model
- ❖ Vector Autoregressive (VAR) Model
- ❖ Recurrent Neural Network
  - ➢ Formulation
  - ➢ Python Implementation

Section IV: Comparison of the statistical and neural network approaches
Section V: Spark Implementation Consideration

Motivation - Why MTSA?

	DATE	USD1MTD156N	USD3MTD156N	USD6MTD156N	USD12MD156N	DGS2	DGS5	DGS10
0	2000-01-04	5.81000	6.04250	6.20875	6.60000	6.30	6.40	6.49
1	2000-01-05	5.79250	6.03000	6.19375	6.57125	6.38	6.51	6.62
2	2000-01-06	5.79125	6.03000	6.19125	6.58000	6.35	6.46	6.57
3	2000-01-07	5.79125	6.03000	6.19125	6.58375	6.31	6.42	6.52
4	2000-01-10	5.78125	6.02625	6.18125	6.57375	6.38	6.49	6.57



Section I

Section I: Time series forecasting problem formulation
Section II: Univariate & Multivariate time series forecasting
Section III: Selected approaches to this problem:
- Autoregressive Integrated Moving Average (ARIMA) Model
- Vector Autoregressive (VAR) Model
- Recurrent Neural Network
  - Formulation
  - Python Implementation

Section IV: Comparison of the statistical and neural network approaches
Section V: Spark Implementation Consideration

Forecasting: Problem Formulation

- Forecasting: predicting the future values of the series using current information set

Forecasting: Problem Formulation

- Forecasting: predicting the **future values** of the series using **current information set**

Forecasting: Problem Formulation

- Forecasting: predicting the **future values** of the series using **current information set**

- **Current information set** consists of current and past values of the series and other "exogenous" series

Time Series Forecasting Requires a Model

Time Series Forecasting Requires Models

$$\hat{y}_{y+H|t} = f(\text{current information set})$$

Time Series Forecasting Requires Models

$$\hat{y}_{y+H|t} = f(\text{current information set})$$

$$= f(y_t, y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{X}_t, \mathbf{X}_{t-1}, \ldots, \mathbf{X}_1)$$

Forecast horizon: H

A statistical model or a machine learning algorithm

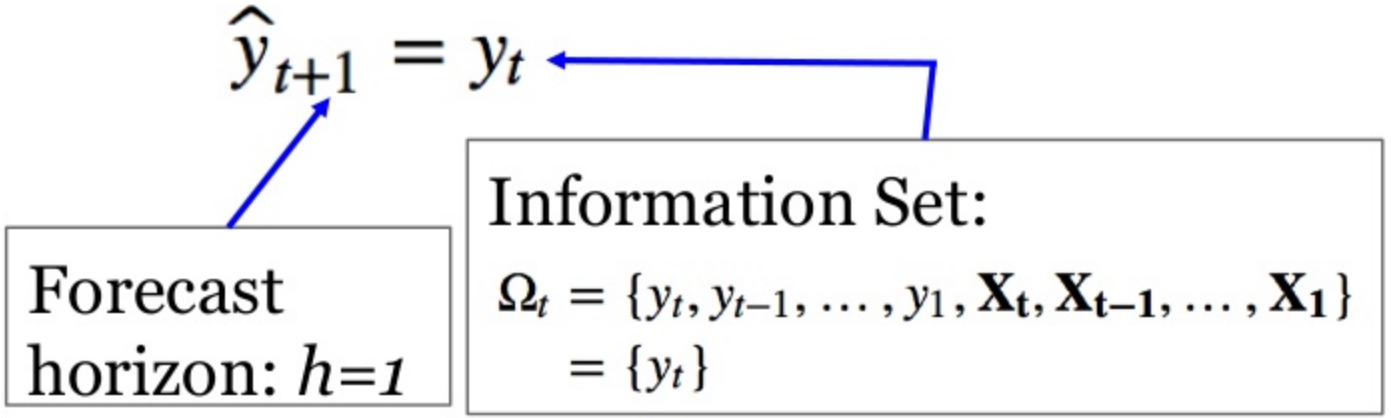
Information Set:

$$\Omega_t = \{y_t, y_{t-1}, y_{t-2}, \ldots, y_1, \mathbf{X}_t, \mathbf{X}_{t-1}, \ldots, \mathbf{X}_1\}$$

A Naïve, Rule-based Model:

A model, *f()*, could be as simple as "a rule" - naive model:

The forecast for tomorrow is the observed value today



However …

$f()$could be a slightly more complicated "model" that utilizes more information from the current information set

"Rolling" Average Model

The forecast for time *t+1* is an average of the observed values from a predefined, past *k* time periods

$$\hat{y}_{t+1} = \frac{1}{k} \sum_{s=t-k}^{t} y_s$$

Forecast horizon: $h=1$

Information Set:

$$\Omega_t = \{y_t, y_{t-1}, \ldots, y_1, \mathbf{X_t}, \mathbf{X_{t-1}}, \ldots, \mathbf{X_1}\}$$

$$= \{y_t, \ldots, y_{t-k}\}$$

More Sophisticated Models

... there are other, much more sophisticated models that can utilize the information set much more efficiently

Section II

Section I: Time series forecasting problem formulation

Section II: Univariate & Multivariate time series forecasting

Section III: Selected approaches to this problem:

- Autoregressive Integrated Moving Average (ARIMA) Model
- Vector Autoregressive (VAR) Model
- Recurrent Neural Network
  - Formulation
  - Python Implementation

Section IV: Comparison of the statistical and neural network approaches

Section V: Spark Implementation Consideration

Univariate Statistical Time Series Models

Statistical relationship is **unidirectional**

$$y_{t+1} \longleftarrow \{y_t, y_{t-1}, \ldots, y_1\}$$

Dynamics of series y

values from its own series

exogenous series

$$y_{t+1} \longleftarrow \{y_t, y_{t-1}, \ldots, y_1, \mathbf{X_t}, \mathbf{X_{t-1}}, \ldots, \mathbf{X_1}\}$$

Autoregressive Moving Average (ARIMA) Model: 1-Minute Recap

$$y_t = a + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p} + \omega_t + \theta_1 \omega_{t-1} + \cdots + \theta_q \omega_{t-q}$$

values from own series

shocks / "error" terms

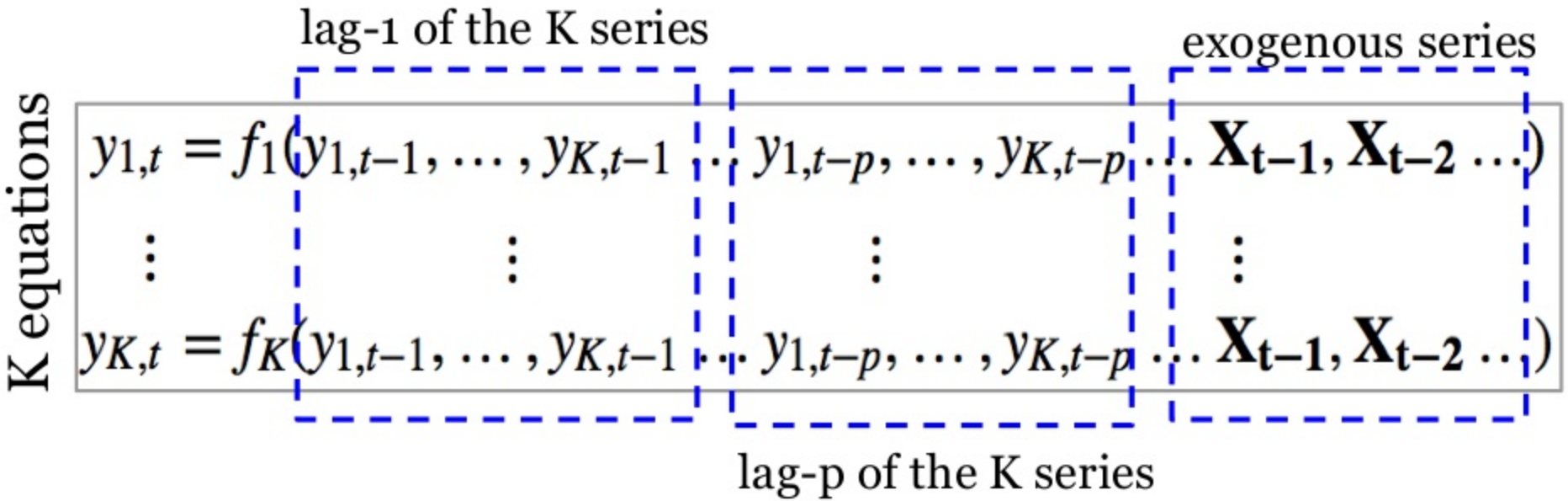
$$\omega_t \sim N(0, \sigma_\omega^2) \quad \forall t$$

It models the dynamics of the series

Autoregressive Integrated Moving Average (ARIMA) Model: 1-Minute Recap

My tutorial at PyData San Francisco 2016

Multivariate Time Series Modeling



Dynamics of each of the series

Interdependence among the series

Section III.A

Section I: Time series forecasting problem formulation

Section II: Univariate & Multivariate time series forecasting

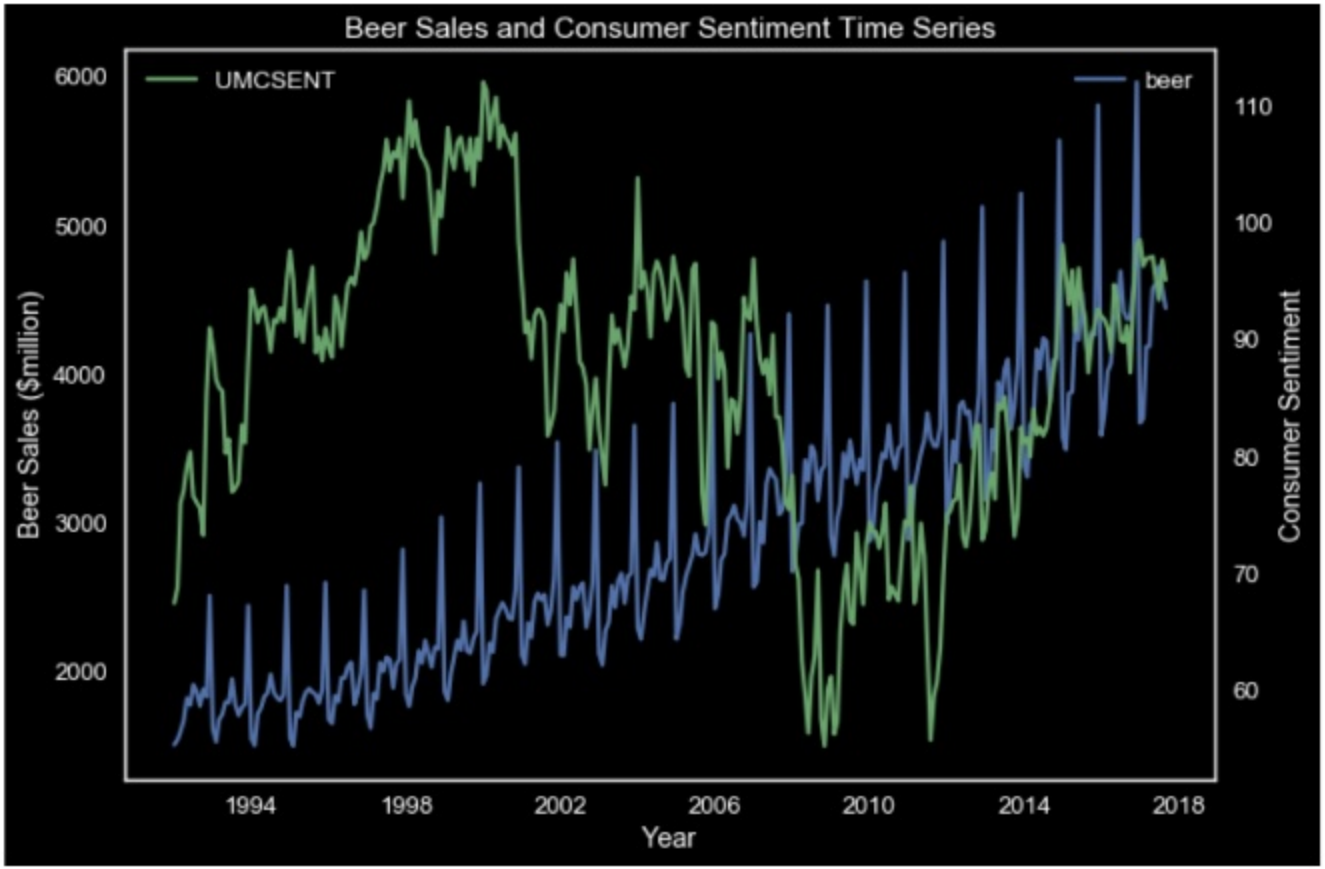
Section III: Selected approaches to this problem:

- ❖ Autoregressive Integrated Moving Average (ARIMA) Model
- ❖ Vector Autoregressive (VAR) Model
  - Recurrent Neural Network
    - Formulation
    - Python Implementation

Section IV: Comparison of the statistical and neural network approaches

Section V: Spark Implementation Consideration

Joint Modeling of Multiple Time Series



Vector Autoregressive (VAR) Models

- a system of linear equations of the K series being modeled
- only applies to stationary series
- non-stationary series can be transformed into stationary ones using simple differencing (note: if the series are not co-integrated, then we can still apply VAR ("VAR in differences"))

Vector Autoregressive (VAR) Model of Order 1

A system of K equations

$$
\begin{aligned}
y_{1,t} &= c_1 + \phi_{11} y_{1,t-1} + \phi_{K2,1} y_{2,t-1} + \cdots + \phi_{1K} y_{K,t-1} + u_{1,t} \\
y_{2,t} &= c_2 + \phi_{21} y_{1,t-1} + \phi_{K2,1} y_{2,t-1} + \cdots + \phi_{2K} y_{K,t-1} + u_{2,t} \\
&\vdots \qquad\qquad\qquad\qquad \vdots \\
y_{K,t} &= c_K + \phi_{K1} y_{1,t-1} + \phi_{K2,1} y_{2,t-1} + \cdots + \phi_{KK} y_{K,t-1} + u_{K,t}
\end{aligned}
$$

Each series is modelled by its own lag as well as other series' lags

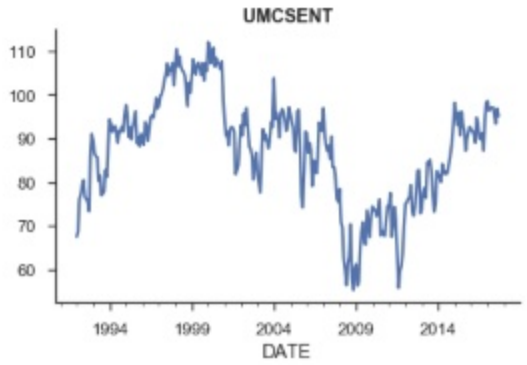
General Steps to Build VAR Model

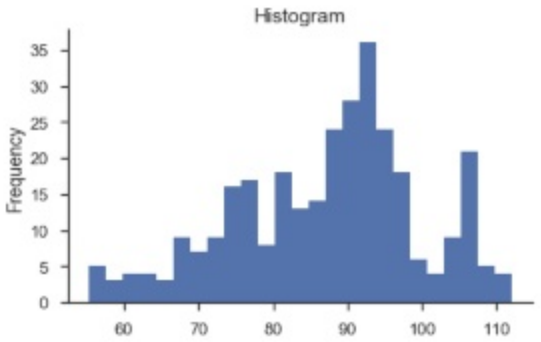
1. Ingest the series
2. Train/validation/test split the series
3. Conduct exploratory time series data analysis on the training set
4. Determine if the series are stationary
5. Transform the series
6. Build a model on the transformed series
7. Model diagnostic
8. Model selection (based on some pre-defined criterion)
9. Conduct forecast using the final, chosen model
10. Inverse-transform the forecast
11. Conduct forecast evaluation

Iterative

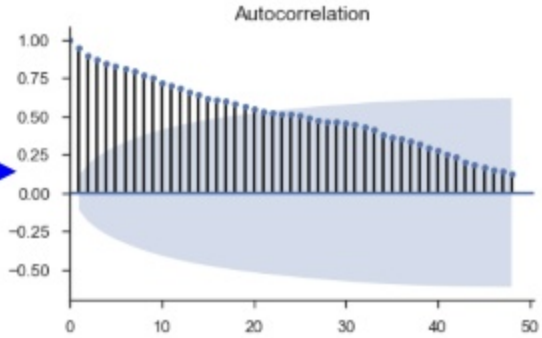
Index of Consumer Sentiment

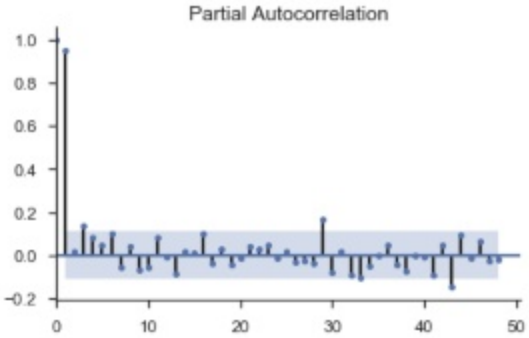
```
def tsplot2(y, title, lags=None, figsize=(12, 8)):
    '''Examine the patterns of ACF and PACF, along with the time series plot and histogram.
    '''
    fig = plt.figure(figsize=figsize)
    layout = (2, 2)
    ts_ax   = plt.subplot2grid(layout, (0, 0))
    hist_ax = plt.subplot2grid(layout, (0, 1))
    acf_ax  = plt.subplot2grid(layout, (1, 0))
    pacf_ax = plt.subplot2grid(layout, (1, 1))

    y.plot(ax=ts_ax)
    ts_ax.set_title(title, fontsize=14, fontweight='bold')
    y.plot(ax=hist_ax, kind='hist', bins=25)
    hist_ax.set_title('Histogram')
    smt.graphics.plot_acf(y, lags=lags, ax=acf_ax)
    smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax)
    [ax.set_xlim(0) for ax in [acf_ax, pacf_ax]]
    sns.despine()
    plt.tight_layout()
    return ts_ax, acf_ax, pacf_ax
```





autocorrelation function (ACF) graph





Partial autocorrelation function (PACF) graph

Series Transformation

Transformation

Applying **first differencing** or **seasonal differencing** to the **log of the series** should make the-above two series stationary:

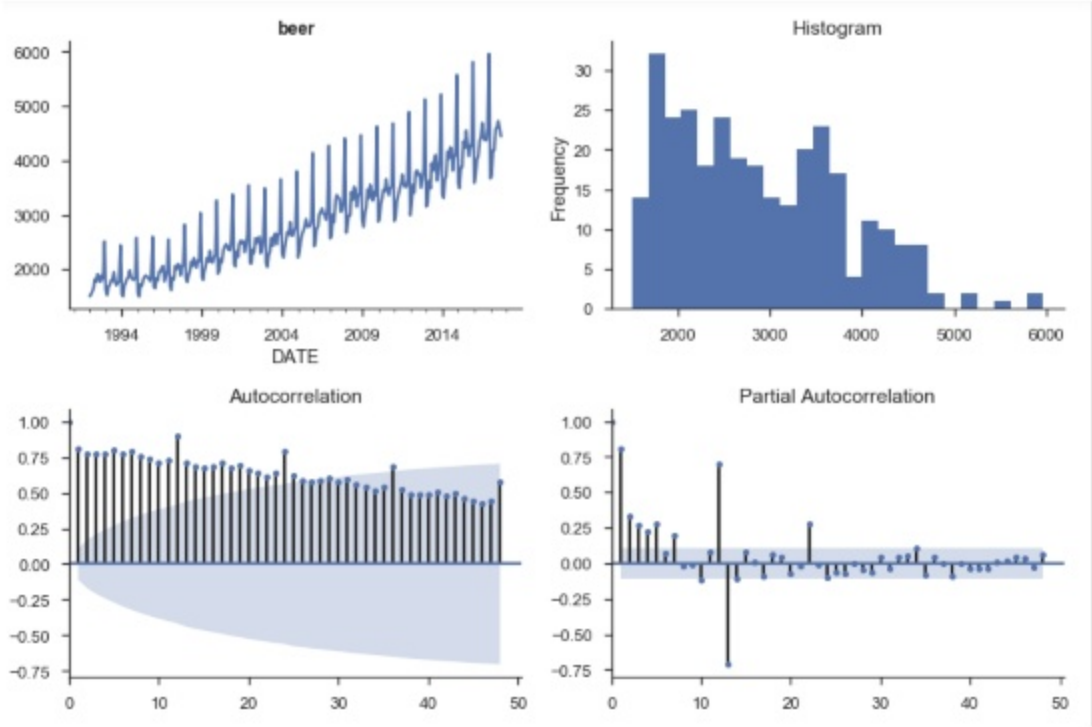
$$log(y_t) - log(y_{t-l}) = log(\frac{y_t}{y_{t-l}})$$

where l is some lag.

- UMCSENT series: $l = 1$
- beer series: $l = 12$

Beer Consumption Series

```
def tsplot2(y, title, lags=None, figsize=(12, 8)):
    '''Examine the patterns of ACF and PACF, along with the time series plot and histogram.
    '''
    fig = plt.figure(figsize=figsize)
    layout = (2, 2)
    ts_ax   = plt.subplot2grid(layout, (0, 0))
    hist_ax = plt.subplot2grid(layout, (0, 1))
    acf_ax  = plt.subplot2grid(layout, (1, 0))
    pacf_ax = plt.subplot2grid(layout, (1, 1))

    y.plot(ax=ts_ax)
    ts_ax.set_title(title, fontsize=14, fontweight='bold')
    y.plot(ax=hist_ax, kind='hist', bins=25)
    hist_ax.set_title('Histogram')
    smt.graphics.plot_acf(y, lags=lags, ax=acf_ax)
    smt.graphics.plot_pacf(y, lags=lags, ax=pacf_ax)
    [ax.set_xlim(0) for ax in [acf_ax, pacf_ax]]
    sns.despine()
    plt.tight_layout()
    return ts_ax, acf_ax, pacf_ax
```



Autocorrelation function

$$\rho_k = \frac{\gamma_k}{\sigma^2} \quad \text{where} \quad \sigma^2 = \gamma_0$$

$$\gamma_k = E[(x_t - \mu)(x_{t+k} - \mu)]$$

$$\gamma_x(s,t) = cov(x_s, x_t) = E[(x_s - \mu_s)(x_t - \mu_t)] \quad \forall s,t$$

Transforming the Series

Take the simple-difference of the natural logarithmic transformation of the series

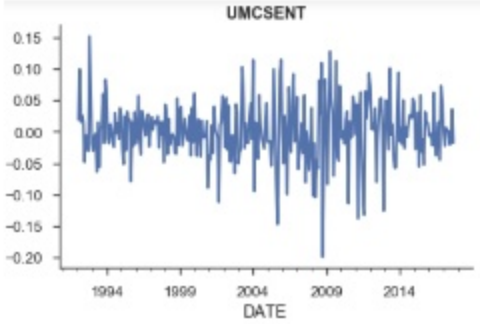
```
series_transformed['UMCSENT'] = np.log(series.iloc[:,0]).diff(1)
series_transformed['beer']    = np.log(series.iloc[:,1]).diff(12)
```

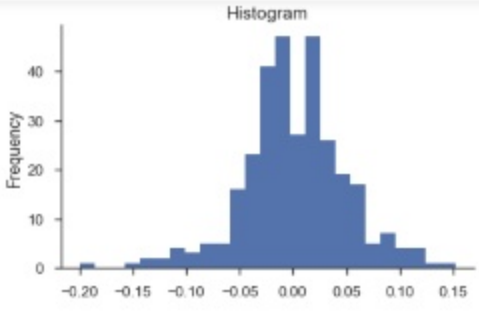
	UMCSENT	beer	UMCSENT	beer
DATE				
1992-01-01	67.5	1509.0	NaN	NaN
1992-02-01	68.8	1541.0	0.019076	NaN
1992-03-01	76.0	1597.0	0.099530	NaN
1992-04-01	77.2	1675.0	0.015666	NaN
1992-05-01	79.2	1822.0	0.025577	NaN
1992-06-01	80.4	1775.0	0.015038	NaN
1992-07-01	76.6	1912.0	-0.048417	NaN
1992-08-01	76.1	1862.0	-0.006549	NaN
1992-09-01	75.6	1770.0	-0.006592	NaN
1992-10-01	73.3	1882.0	-0.030896	NaN
1992-11-01	85.3	1831.0	0.151614	NaN
1992-12-01	91.0	2511.0	0.064685	NaN
1993-01-01	89.3	1614.0	-0.018858	0.067268
1993-02-01	86.6	1529.0	-0.030702	-0.007818
1993-03-01	85.9	1678.0	-0.008116	0.049476

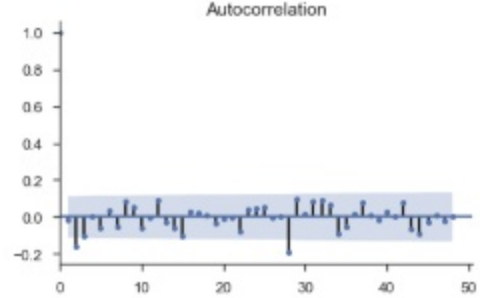
note: difference-transformation generates missing values

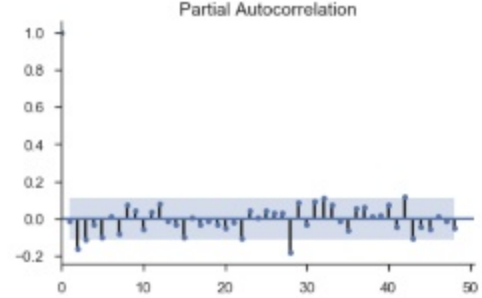
Transformed Series

Consumer Sentiment

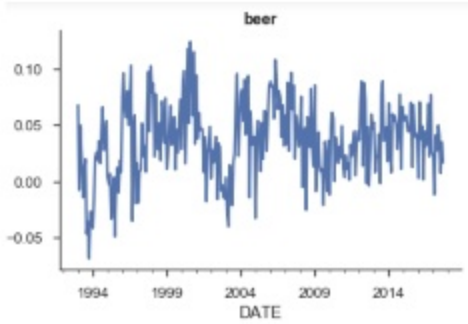


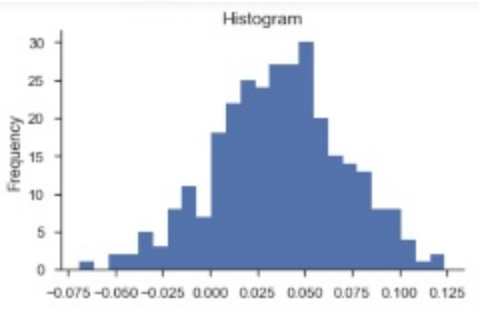


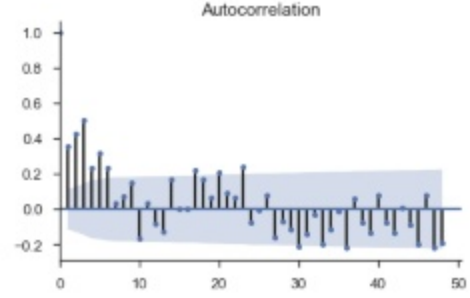


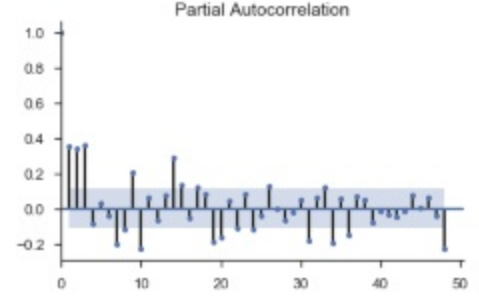


Beer Consumption









VAR Model Proposed

Is the method we propose capable of answering the following questions?

- What are the dynamic properties of these series? Own lagged coefficients
- How are these series interact, if at all? Cross-series lagged coefficients

$$y_{1,t} = c_1 + \phi_{11} y_{1,t-1} + \phi_{12} y_{2,t-1} + \phi_{13} y_{1,t-2} + \phi_{14} y_{2,t-2} + \phi_{15} y_{1,t-3} + \phi_{16} y_{2,t-3} + u_{1,t}$$

$$y_{2,t} = c_2 + \phi_{21} y_{1,t-1} + \phi_{22} y_{2,t-1} + \phi_{23} y_{1,t-2} + \phi_{24} y_{2,t-2} + \phi_{25} y_{1,t-3} + \phi_{26} y_{2,t-3} + u_{2,t}$$

VAR Model Estimation and Output

```
model = sm.tsa.VARMAX(y_train, order=(3,0), trend='c')
model_result = model.fit(maxiter=1000, disp=False)
print(model_result.summary())
```

```
                            Statespace Model Results
==============================================================================
Dep. Variable:     ['UMCSENT', 'beer']   No. Observations:                  294
Model:                          VAR(3)   Log Likelihood                1124.934
                          + intercept   AIC                          -2215.867
Date:                 Thu, 30 Nov 2017   BIC                          -2153.246
Time:                         11:25:18   HQIC                         -2190.790
Sample:                     01-01-1993
                          - 06-01-2017
Covariance Type:                  opg
===================================================================================
Ljung-Box (Q):                52.72, 181.87   Jarque-Bera (JB):           21.96, 1.78
Prob(Q):                       0.09, 0.00   Prob(JB):                    0.00, 0.41
Heteroskedasticity (H):        2.26, 0.62   Skew:                       -0.38, -0.19
Prob(H) (two-sided):           0.00, 0.02   Kurtosis:                    4.11, 3.03
```

VAR Model Output - Estimated Coefficients

Results for equation UMCSENT

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0060	0.005	1.266	0.206	-0.003	0.015
L1.UMCSENT	-0.0551	0.051	-1.089	0.276	-0.154	0.044
L1.beer	-0.1338	0.105	-1.274	0.203	-0.340	0.072
L2.UMCSENT	-0.1654	0.060	-2.774	0.006	-0.282	-0.049
L2.beer	0.0174	0.096	0.182	0.856	-0.171	0.205
L3.UMCSENT	-0.1218	0.054	-2.247	0.025	-0.228	-0.016
L3.beer	-0.0398	0.089	-0.446	0.656	-0.215	0.135

Results for equation beer

	coef	std err	z	P>|z|	[0.025	0.975]
const	0.0097	0.003	3.377	0.001	0.004	0.015
L1.UMCSENT	0.0559	0.041	1.375	0.169	-0.024	0.136
L1.beer	0.1060	0.055	1.920	0.055	-0.002	0.214
L2.UMCSENT	0.0292	0.038	0.764	0.445	-0.046	0.104
L2.beer	0.2616	0.056	4.674	0.000	0.152	0.371
L3.UMCSENT	0.0254	0.036	0.711	0.477	-0.045	0.096
L3.beer	0.3698	0.057	6.507	0.000	0.258	0.481

VAR Model Output - Var-Covar Matrix

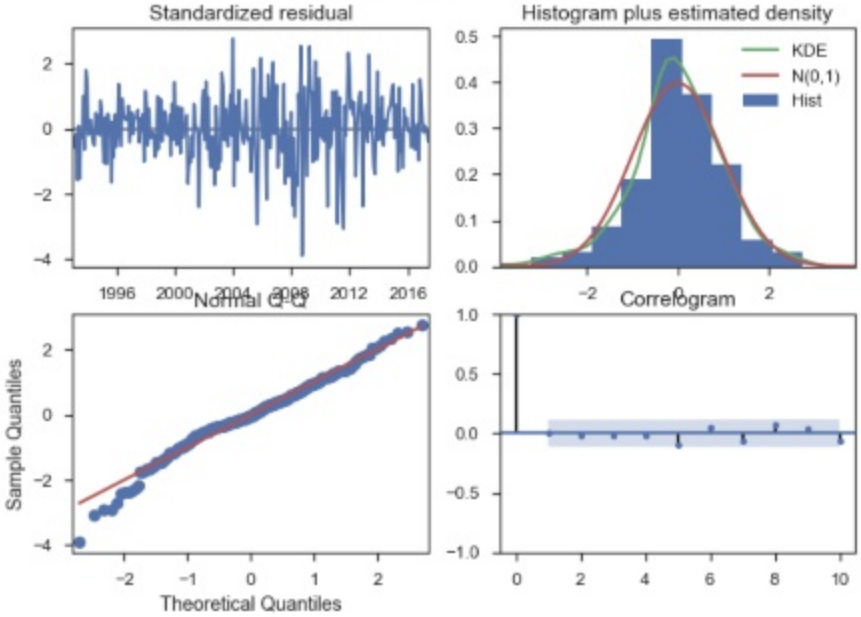
Error covariance matrix

	coef	std err	z	P>|z|	[0.025	0.975]
sqrt.var.UMCSENT	0.0462	0.002	27.616	0.000	0.043	0.049
sqrt.cov.UMCSENT.beer	0.0004	0.002	0.211	0.833	-0.003	0.004
sqrt.var.beer	0.0276	0.001	23.581	0.000	0.025	0.030

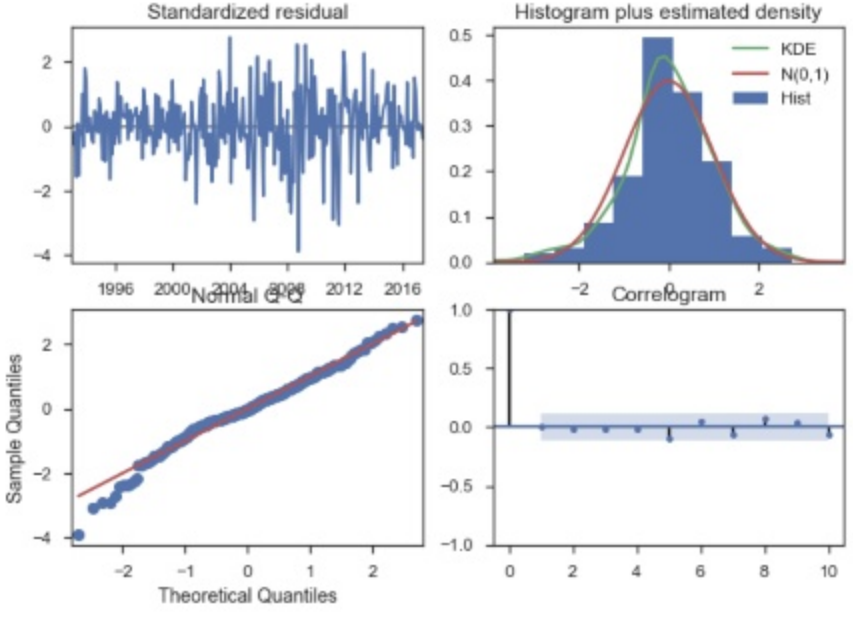
VAR Model Diagnostic

```
model = sm.tsa.VARMAX(y_train, order=(3,0), trend='c')
model_result = model.fit(maxiter=1000, disp=False)
model_result.plot_diagnostics()
```

UMCSENT



Beer



VAR Model Selection

Model selection, in the case of VAR(p), is the choice of the order and the specification of each equation

Information criterion can be used for model selection:

```
aic = []
for i in range(5):
    i += 1
    model = sm.tsa.VARMAX(y_train, order=(i,0), trend='c')
    model_result = model.fit(maxiter=1000, disp=False)
    print('Order =', i)
    print('AIC: ', model_result.aic)
    print('BIC: ', model_result.bic)
    print('HQIC: ', model_result.hqic)
```

VAR Model - Inverse Transform

Don't forget to inverse-transform the forecasted series!

This is equivalent to $log(y_t) - log(y_{t-12}) = log(\frac{y_t}{y_{t-12}})$

Define $z = log(y_t) - log(y_{t-12})$

Then,

$$\begin{aligned} log(y_t) &= z + log(y_{t-12}) \\ y_t &= e^{z+log(y_{t-12})} \\ &= e^z(y_{t-12}) \end{aligned}$$

So, we have forecast

$$y_{T+s} = e^z(y_{(t-12)+s})$$

where $s > 1$

VAR Model - Forecast Using the Model

The Forecast Equation:

$$\hat{y}_{1,T+1|T} = \hat{c}_1 + \widehat{\phi}_{11} y_{1,T} + \widehat{\phi}_{12} y_{2,T} + \widehat{\phi}_{13} y_{1,T-1} + \widehat{\phi}_{14} y_{2,T-1} + \widehat{\phi}_{15} y_{1,T-2} + \widehat{\phi}_{16} y_{2,T-2}$$

$$\hat{y}_{2,T+1|T} = \hat{c}_1 + \widehat{\phi}_{21} y_{1,T} + \widehat{\phi}_{22} y_{2,T} + \widehat{\phi}_{23} y_{1,T-1} + \widehat{\phi}_{24} y_{2,T-1} + \widehat{\phi}_{25} y_{1,T-2} + \widehat{\phi}_{26} y_{2,T-2}$$

VAR Model Forecast

$$RMSE = \sqrt{\frac{1}{L}\sum_{l=1}^{L}\left(y_{T+l} - \hat{y}_{T+l}\right)^2}$$

where T is the last observation period and l is the lag

```
from math import sqrt
from sklearn.metrics import mean_squared_error

VAR_forecast_beer = np.exp(z['beer'])*series['beer'][-3:]
VAR_forecast_UMCSENT = np.exp(z['UMCSENT'])*series['UMCSENT'][-3:]

rmse_beer = sqrt(mean_squared_error(series['beer'][-3:], VAR_forecast_beer))
rmse_UMCSENT = sqrt(mean_squared_error(series['UMCSENT'][-3:], VAR_forecast_UMCSENT))
```

```
UMSCENT - Test RMSE: 0.210
Beer    - Test RMSE: 180.737
```

What do the result mean in this context?

Don't forget to put the result in the existing context!

```
UMSCENT - Test RMSE: 0.210
Beer    - Test RMSE: 180.737
```

DATE	UMCSENT	beer
2017-07-01	93.4	4726.0
2017-08-01	96.8	4577.0
2017-09-01	95.1	4445.0

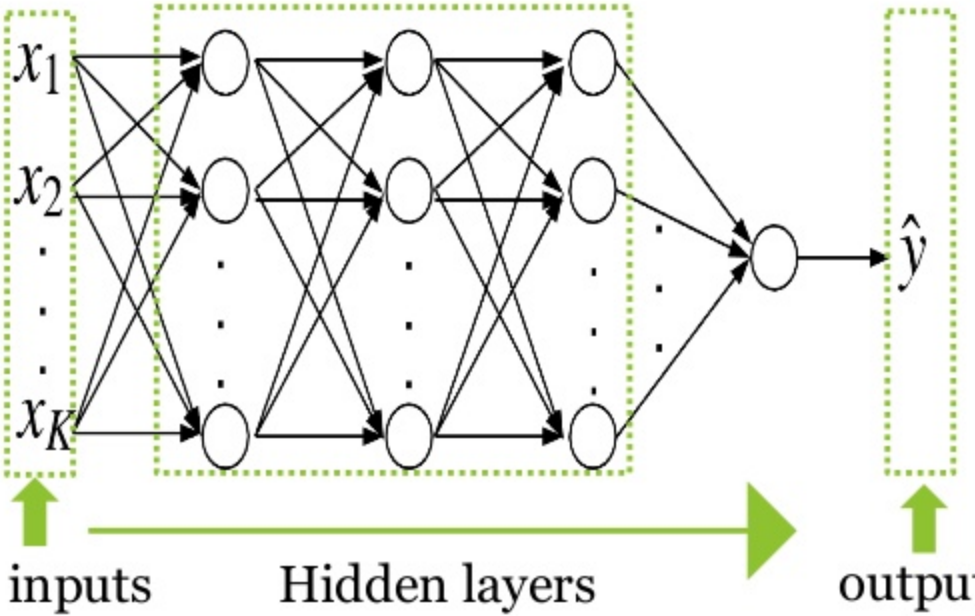
```
UMSCENT - Percentage Error relative to the mean: 0.22
Beer    - Percentage Error relative to the mean: 3.94
```

Section III.B

Section I: Time series forecasting problem formulation
Section II: Univariate & Multivariate time series forecasting
Section III: Selected approaches to this problem:
- Autoregressive Integrated Moving Average (ARIMA) Model
- Vector Autoregressive (VAR) Model
- ❖ Recurrent Neural Network
  - Formulation
  - Python Implementation

Section IV: Comparison of the statistical and neural network approaches
Section V: Spark Implementation Consideration

Feed-Forward Network with a Single Output



- Information does not account for time ordering
- inputs are processed independently
- no “device” to keep the past information

Network architecture does not have "memory" built in

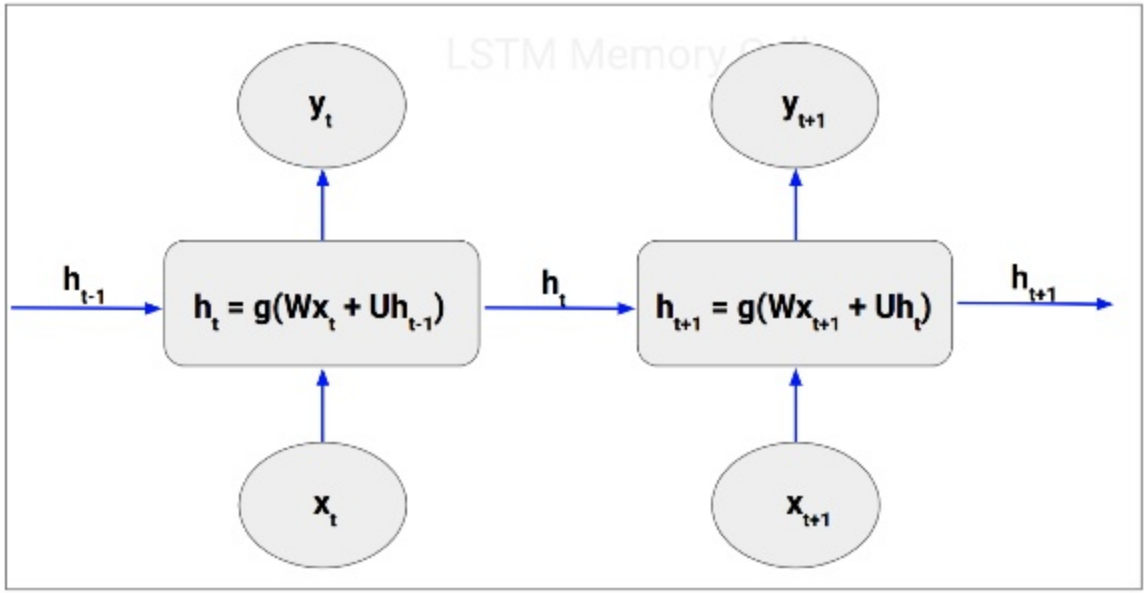
Recurrent Neural Network (RNN)

A network architecture that can

- retain past information
- track the state of the world, and
- update the state of the world as the network moves forward

Handles variable-length sequence by having a recurrent hidden state whose activation at each time is dependent on that of the previous time.

Standard Recurrent Neural Network (RNN)



Limitation of Vanilla RNN Architecture

Exploding (and vanishing) gradient problems
(Sepp Hochreiter, 1991 Diploma Thesis)

Long Short Term Memory (LSTM) Network

LONG SHORT-TERM MEMORY

NEURAL COMPUTATION 9(8):1735 1780, 1997

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

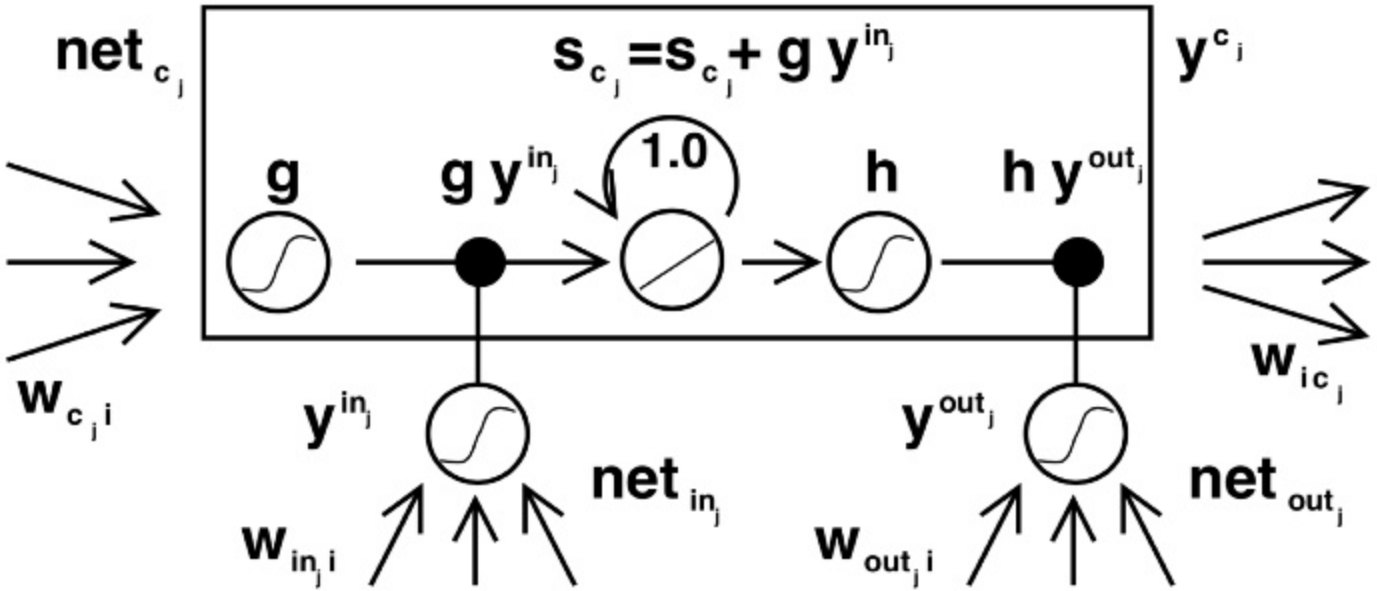
Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Abstract

Learning to store information over extended time intervals via recurrent backpropagation takes a very long time, mostly due to insufficient, decaying error back flow. We briefly review Hochreiter's 1991 analysis of this problem, then address it by introducing a novel, efficient, gradient-based method called "Long Short-Term Memory" (LSTM). Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete time steps by enforcing *constant* error flow through "constant error carrousels" within special units. Multiplicative gate units learn to open and close access to the constant error flow. LSTM is local in space and time; its computational complexity per time step and weight is $O(1)$. Our experiments with artificial data involve local, distributed, real-valued, and noisy pattern representations. In comparisons with RTRL, BPTT, Recurrent Cascade-Correlation, Elman nets, and Neural Sequence Chunking, LSTM leads to many more successful runs, and learns much faster. LSTM also solves complex, artificial long time lag tasks that have never been solved by previous recurrent network algorithms.

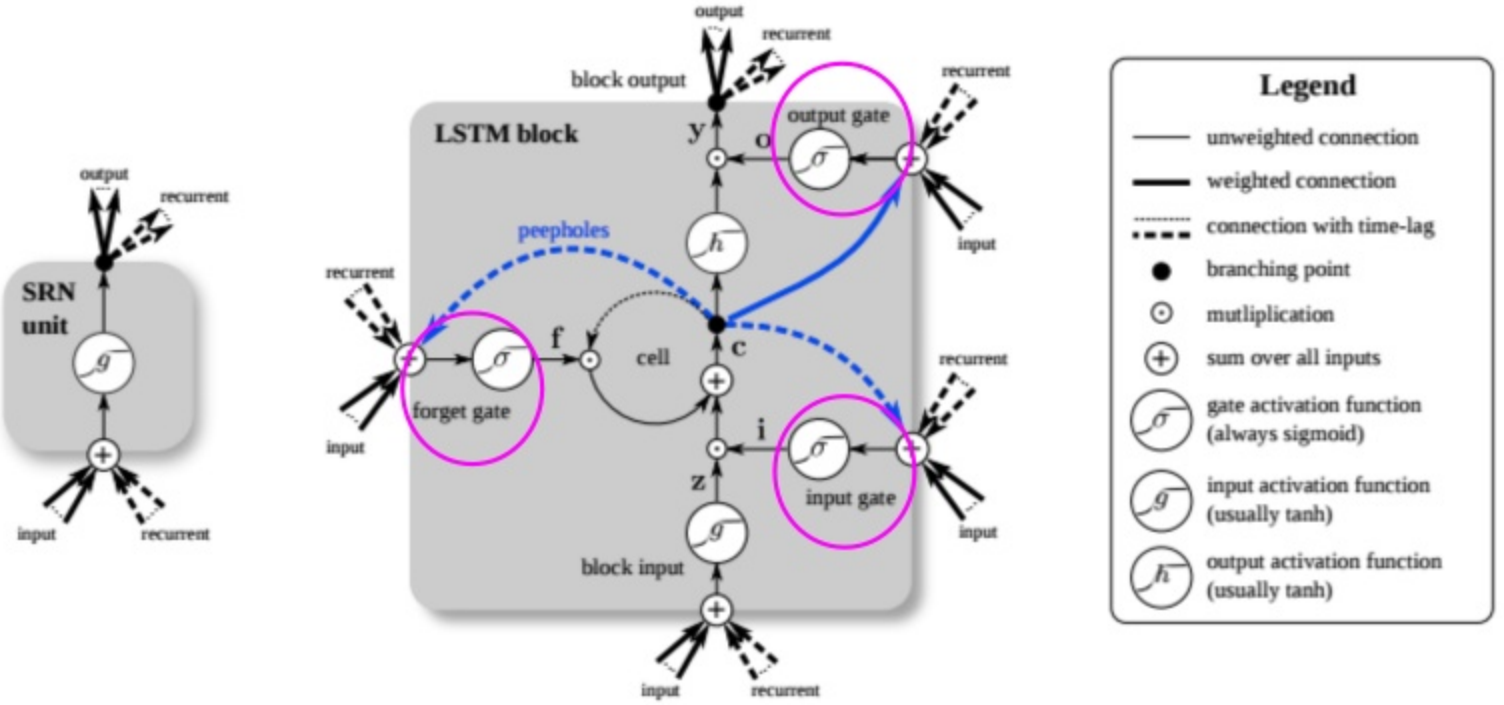
LSTM: Hochreiter and Schmidhuber (1997)

The architecture of memory cells and gate units from the original Hochreiter and Schmidhuber (1997) paper



Long Short Term Memory (LSTM) Network

Another representation of the architecture of memory cells and gate units: Greff, Srivastava, Koutnık, Steunebrink, Schmidhuber (2016)

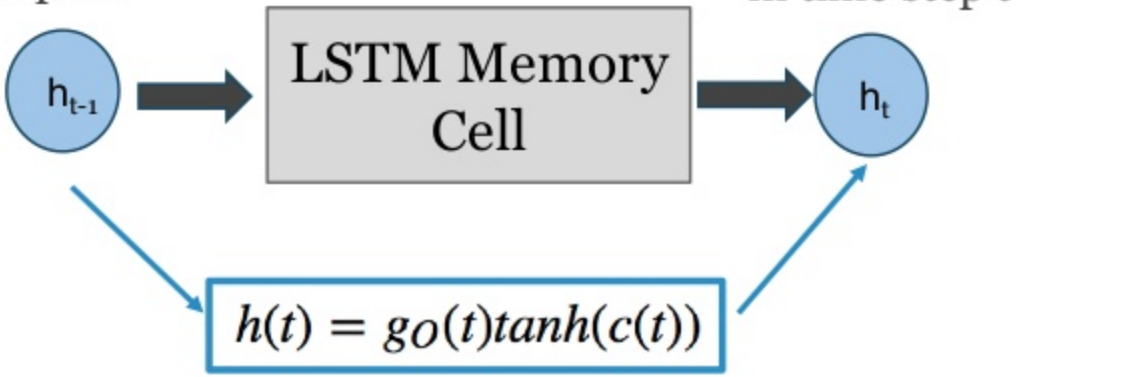


LSTM: An Overview

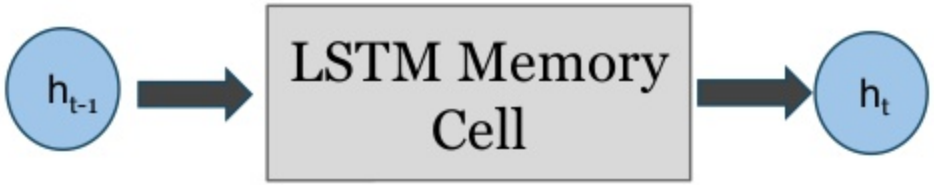
Use memory cells and gated units for information flow

hidden state (value from activation function) in time step t-1

hidden state (value from activation function) in time step t



LSTM: An Overview



hidden state memory cell (state)

$$h(t) = g_O(t)tanh(c(t))$$

$$tanh(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Output gate

$$g_O(t) = \sigma\left(W_O \cdot [h(t-1), x(t)] + b_O\right)$$

$$c(t) = g_F(t)c(t-1) + g_I(t)\tilde{c}(t) \leftarrow \tilde{c}(t) = tanh\left(W_c \cdot [h(t-1), x(t)] + b_c\right)$$

Forget gate

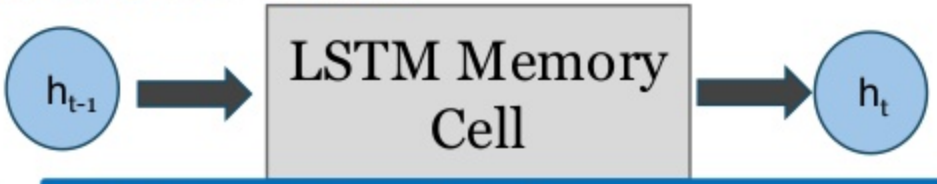
$$g_F(t) = \sigma\left(W_F \cdot [h(t-I), x(t)] + b_F\right)$$

Input gate

$$g_I(t) = \sigma\left(W_I \cdot [h(t-I), x(t)] + b_I\right)$$

Training uses Backward Propagation Through Time (BPTT)

LSTM: An Overview



hidden state(t)

$$h(t) = g_O(t)tanh(c(t))$$

memory cell (t)

$$c(t) = g_F(t)c(t-1) + g_I(t)\tilde{c(t)}$$

Candidate memory cell (t)

$$\tilde{c(t)} = tanh\left(W_c \cdot [h(t-1), x(t)] + b_c\right)$$

Output gate

$$g_O(t) = \sigma\left(W_O \cdot [h(t-1), x(t)] + b_O\right)$$

Input gate

$$g_I(t) = \sigma\left(W_I \cdot [h(t-I), x(t)] + b_I\right)$$

Forget gate

$$g_F(t) = \sigma\left(W_F \cdot [h(t-I), x(t)] + b_F\right)$$

Training uses Backward Propagation Through Time (BPTT)

Implementation in Keras

Some steps to highlight:

- Formulate the series for a RNN supervised learning regression problem (i.e. (Define target and input tensors))
- Scale all the series
- Split the series for training/development/testing
- Reshape the series for (Keras) RNN implementation
- Define the (initial) architecture of the LSTM Model
  - Define a network of layers that maps your inputs to your targets and the complexity of each layer (i.e. number of memory cells)
  - Configure the learning process by picking a loss function, an optimizer, and metrics to monitor
- Produce the forecasts and then reverse-scale the forecasted series
- Calculate loss metrics (e.g. RMSE, MAE)

Note that stationarity, as defined previously, is not a requirement

Continue with the Same Example

LSTM Architecture Design, Training, Evaluation

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# Design the network architecture
model = Sequential()
model.add(LSTM(60,
               dropout=0.1,
               recurrent_dropout=0.2,
               return_sequences = True,
               input_shape=(n_lookback,X_scaled_train_reshape.shape[2])))
model.add(LSTM(36))
model.add(Dense(X_scaled_train_reshape.shape[2]))
model.compile(loss='mae', optimizer='RMSprop')

# Model Training
n_epochs=500
batchSize = 40
model.fit(X_scaled_train_reshape, y_scaled_train,epochs=n_epochs,
          batch_size=batchSize, verbose=0, shuffle=False)

# make a prediction
yhat_scale = model.predict(X_scaled_test_reshape)

# Inverse-scaling for forecast
inv_yhat = np.concatenate((X_scaled_test, yhat_scale), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)

# Model Evaluation
from math import sqrt
from sklearn.metrics import mean_squared_error
print('Test RMSE: %.3f' % sqrt(mean_squared_error(y_test[:,0], inv_yhat[:,0])))
print('Test RMSE: %.3f' % sqrt(mean_squared_error(y_test[:,1], inv_yhat[:,1])))
```

LSTM: Forecast Results

```
# calculate RMSE
from math import sqrt
from sklearn.metrics import mean_squared_error

print('Test RMSE: %.3f' % sqrt(mean_squared_error(y_test[:,0], inv_yhat[:,0])))
print('Test RMSE: %.3f' % sqrt(mean_squared_error(y_test[:,1], inv_yhat[:,1])))
```

```
Test RMSE: 1.874
Test RMSE: 77.214
```

DATE	UMCSENT	beer
2017-07-01	93.4	4726.0
2017-08-01	96.8	4577.0
2017-09-01	95.1	4445.0

```
print(str(round((sqrt(mean_squared_error(y_test[:,0], inv_yhat[:,0]))/y_test[:,0].mean())*100, 2)) +'%')
print(str(round((sqrt(mean_squared_error(y_test[:,1], inv_yhat[:,1]))/y_test[:,1].mean())*100, 2)) + '%')
```

executed in 8ms, finished 12:41:54 2018-10-04

```
1.97%
1.68%
```

UMSCENT

Beer

Section IV

Section I: Time series forecasting problem formulation
Section II: Univariate & Multivariate time series forecasting
Section III: Selected approaches to this problem:
- Autoregressive Integrated Moving Average (ARIMA) Model
- Vector Autoregressive (VAR) Model
- Recurrent Neural Network
  - Formulation
  - Python Implementation

Section IV: Comparison of the statistical and neural network approaches
Section V: Spark Implementation Consideration

VAR vs. LSTM: Data Type

VAR	LSTM
macroeconomic time series, financial time series, business time series, and other numeric series	DNA sequences, images, voice sequences, texts, all the numeric time series (that can be modeled by VAR)

VAR vs. LSTM: Parametric form

VAR	LSTM
A linear system of equations - highly parameterized (can be formulated in the general state space model)	Layer(s) of many non-linear transformations

VAR vs. LSTM: Stationarity Requirement

VAR	LSTM
• applied to stationary time series only • its variant (e.g. Vector Error Correction Model) can be applied to co-integrated series	• stationarity not a requirement

VAR vs. LSTM: Model Implementation

VAR	LSTM
• data preprocessing is straight-forward • model specification is relative straight-forward, model training time is fast	• data preprocessing is a lot more involved • network architecture design, model training and hyperparameter tuning requires much more efforts

Section V

Section I: Time series forecasting problem formulation
Section II: Univariate & Multivariate time series forecasting
Section III: Selected approaches to this problem:
- Autoregressive Integrated Moving Average (ARIMA) Model
- Vector Autoregressive (VAR) Model
- Recurrent Neural Network
  - Formulation
  - Python Implementation

Section IV: Comparison of the statistical and neural network approaches
Section V: Spark Implementation Consideration

Implementation using Spark

One may think it is straightforward to use Spark for all the steps (outlined above) up to model training, such as ...

```
series = sqlContext.read.format("csv").load(path = "...", header = True, inferSchema = True)
from pyspark.sql.functions import ...
from pyspark.sql ...

w = Window.partitionBy().orderBy(col("DATE"))
lag_period = 5
for i in range(lag_period):
  j = i +1
  for cols in series.columns:
    series = series.withColumn(cols, lag(cols, j).over(w).alias('%s(t-%d)' %(cols, j)))
# Convert ... columns to feature vectors
vectorAssembler = VectorAssembler(inputCols=["UMCSENT","beer"], outputCol="features" )
```

Wait ... Not So Fast!

Ordering in the series matter

Flint: A Time Series Library for Spark

Need time-series aware libraries whose operations preserve the natural order of the series, such as **Flint**, open sourced by TwoSigma (https://github.com/twosigma/flint)

- can handle a variety of datafile types, manipulate and aggregate time series, and provide time-series-specific functions, such as (time-series) join, (time-series) filtering, time-based windowing, cycles, and intervalizing)

Example: Defining a moving average of the beer consumption series

```
@ts.flint.udf(DoubleType())
def movingAverage(window):
    nrows = len(window)
    if nrows == 0:
        return 0
    return sum(row.beer for row in window) / nrows
```

A Few Words on TSA Workflow in Spark

Load, pre-process, analyze, transform series using time-series aware libraries

Conduct Hyperparameter Tuning in a distributed model and use Spark ML Pipeline

```
from pyspark.ml.tuning import ParamGridBuilder

VAR_model = sm.tsa.VARMAX(y_train, order=(p=1,q=0), trend='c')
model_result = model.fit(maxiter=1000, disp=False)

paramGrid = ParamGridBuilder()\
    .addGrid(VAR_model.order, [1, 2, 3, 4, 5]) \
    .addGrid(VAR_model.trend, ['nc','c'])
    .build()
```

Thank You!

jyau@berkeley.edu