On deep machine learning & time series models: A case study with the use of Keras

Presentation · June 2017

DOI: 10.13140/RG.2.2.34543.94888

1 author:

Carlin Chun-fai Chu
The Open University of Hong Kong

On deep machine learning & time series models: A case study with the use of Keras

1st International Conference on Econometrics and Statistics
Session EO256: Business analytics

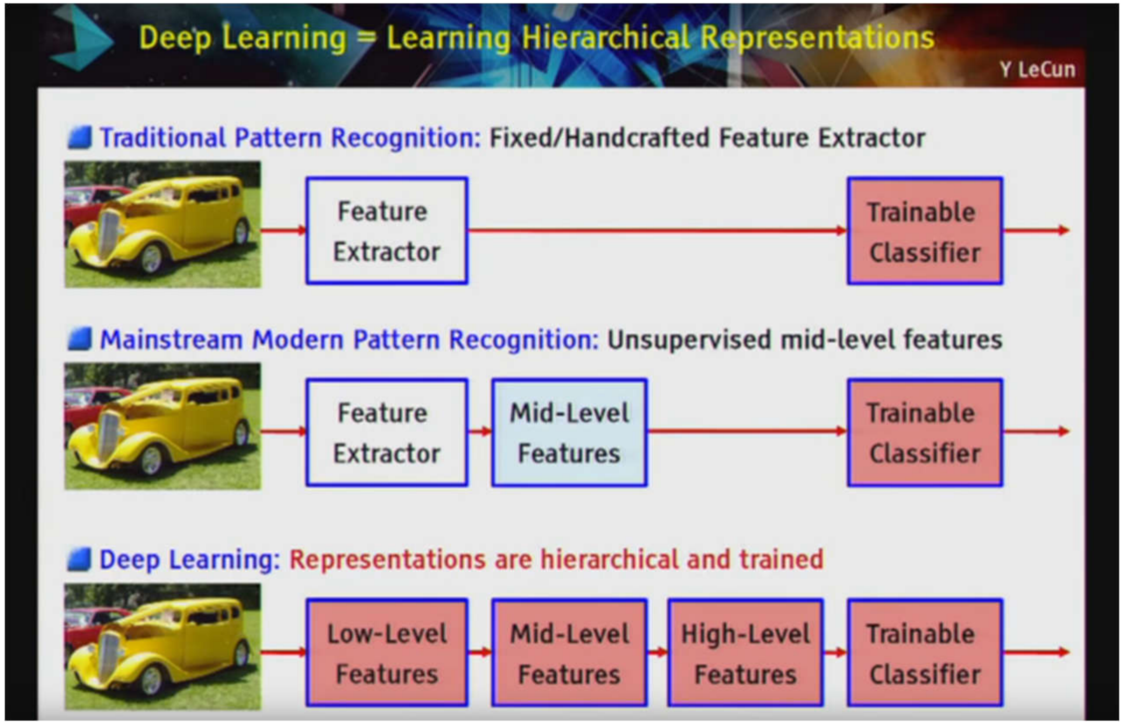
Carlin Chu - The Open University of Hong Kong (Hong Kong)
ccfchu@ouhk.edu.hk

Content

- Why deep learning ?
- What is LSTM ?
- What is Keras ? Characteristics of Keras
- Suggested steps for LSTM coding
- Example codes
- Bollerslev's time series model
- Work-in-progress

Why deep learning ?

- Deep learning
  - Artificial Neural Networks with > 1 hidden layer
  - Involves a lot of data for training
  - Different level of abstractions



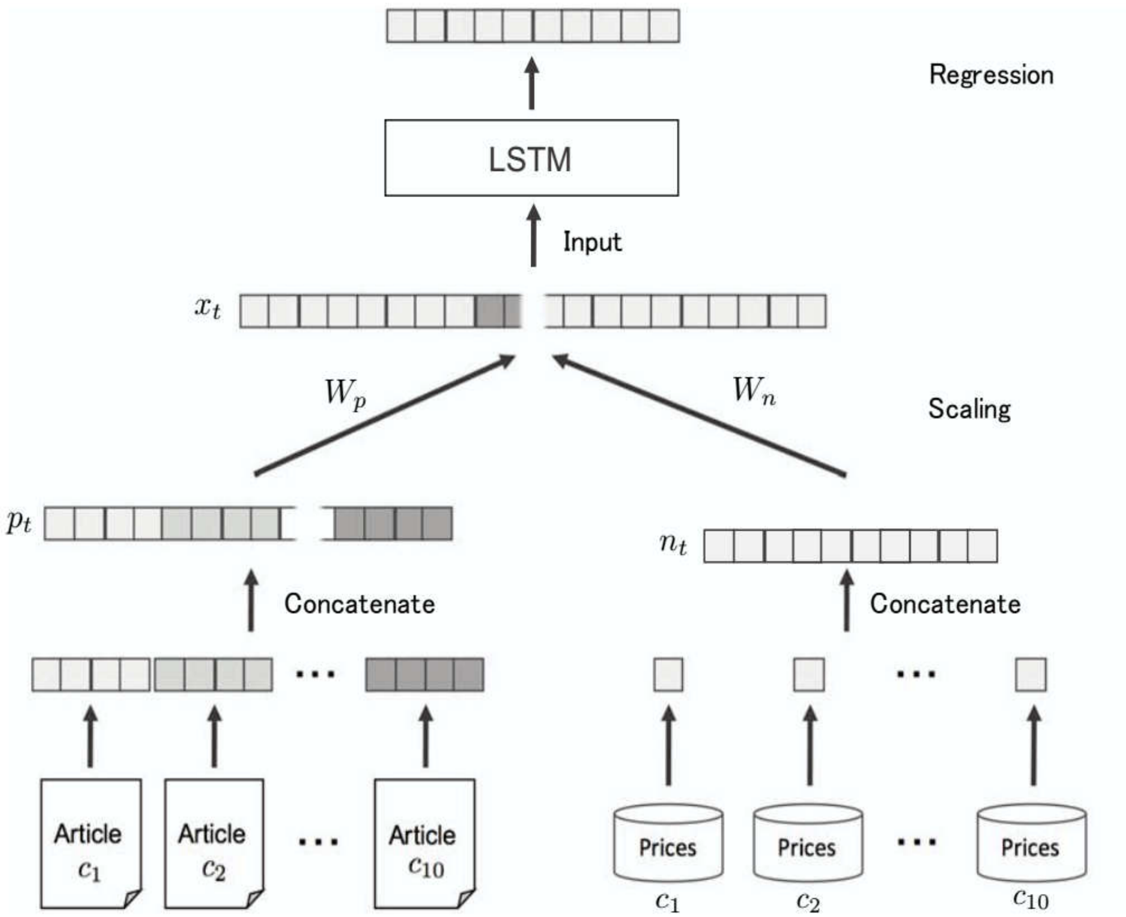
The picture is extracted from: http://machinelearningmastery.com/what-is-deep-learning/
Why Deep Learning? (Slide by Yann LeCun)

Applications of deep learning

- Ancient China board game: GO
- Number of moves > total number of atoms in the world
  - Exhaustive search is not possible.
- **Deep Neural Network & Advanced tree search & Reinforcement learning**

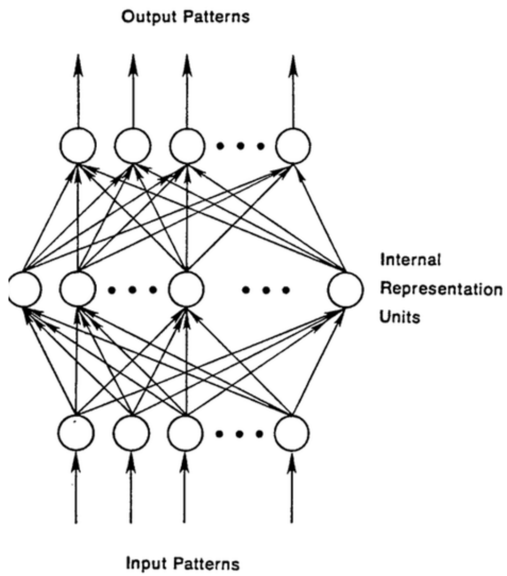
Akita et al. (2016)
News (Textual) + Stock Price (Numerical)

- Info from News article stream + Daily open price ➔ Close price
- Prediction for 10 company using input dimension=1000
- Recurrent NN: Long Short-Term Memory (LSTM)



- ***Deep Learning for Stock Prediction Using Numerical and Textual Information (Akita et al. 2016)***

What is LSTM ?

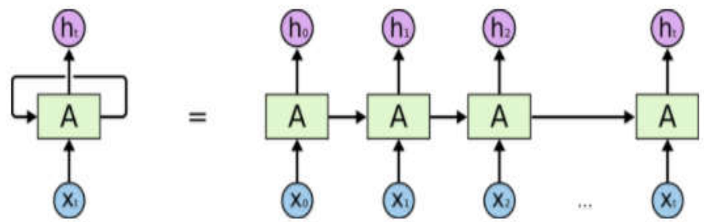


Traditional Artificial Neural Network (ANN)

- no notion of time ordering
- map the current input feature(s) to the predicted target variable(s)

Recurrent Neural Network (RNN)

- with 'loops' which allow information to persist.
- multiple copies of the same network, each passing a message to a successor.



An unrolled recurrent neural network.

Long Short Term Memory network (LSTM)

- special kind of RNN with
- Adaptive forget gate, throw away information
- Keep information with time gaps of unknown/different size(s)

A short review can be found on 'A Beginner's Guide to Recurrent Networks and LSTMs'

- https://deeplearning4j.org/lstm.html

Characteristics of Keras

- high-level neural networks API, written in Python

- run on top of either TensorFlow / Theano / CNTK

- utilize both CPU and GPU

- Part of AlphaGo is written on TensorFlow (for distributed computing)

Backend: Theano or TensorFlow ?

- Which one is better ?
  - Distributed setting/newer software
    → TensorFlow
  - Recurrent network / legacy application
    → Theano

François Chollet, Deep learning researcher at Google, author of Keras
Answered Aug 16, 2016

It depends on what you are doing. For models that involve recurrent networks, then I would recommend Theano for performance reasons. And if you need to run your models in a distributed setting, then I would recommend TensorFlow. Otherwise, use whichever one seems to be the fastest for the model you are using.

In general, TensorFlow has been making very fast progress on the performance front (although RNN performance still leaves much to be desired) and it will eventually replace Theano as the default Keras backend. However, we are not quite there just yet.

- How to switch the backend ?
  - Locate the .json file
  - Change the 'backend' field

Switching from one backend to another

If you have run Keras at least once, you will find the Keras configuration file at:

`$HOME/.keras/keras.json`

The default configuration file looks like this:

```
{
    "image_data_format": "channels_last",
    "epsilon": 1e-07,
    "floatx": "float32",
    "backend": "tensorflow"
}
```

Simply change the field `backend` to `"theano"`, `"tensorflow"`, or `"cntk"`, and Keras will use the new configuration next time you run any Keras code.

Suggested steps for LSTM coding

1. Normalize the data (Transformation)
2. Data preparation to a 3D dataset
3. Model specification
4. Model training (tackle over-fit issue)
5. Prediction
6. Inverse transformation

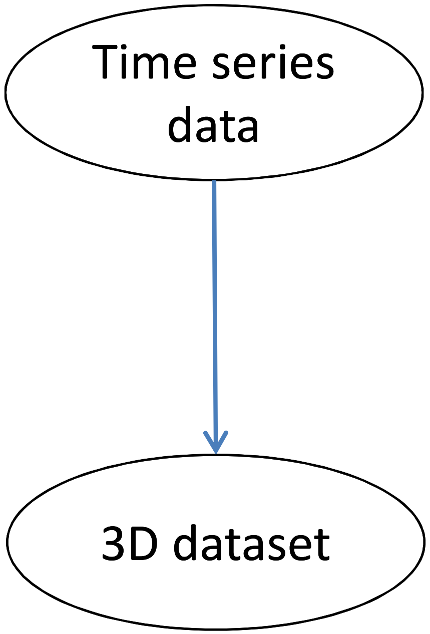
Suggested steps for LSTM coding (1)

- Normalize the data (Transformation)
- Transformation of input and target variables
  - tends to make the training process better behaved by improving the numerical condition of the optimization problem
  - ensuring that various default values involved in initialization and termination are appropriate.
  - ftp://ftp.sas.com/pub/neural/illcond/illcond.html

```
from sklearn.preprocessing import MinMaxScaler

# normalize the dataset
scaler = MinMaxScaler(feature_range=(-1, 1))
normalized_data = scaler.fit_transform(input_data)
```

Suggested steps for LSTM coding (2)

- Data preparation to a 3D dataset



a) Padding original data series with duplicated/repeated values

b) Separate the input feature data and target value data

c) Reshape the padded input data to a 3 dimensional dataset
[samples, time steps, features]

Suggested steps for LSTM coding (2)

- Data preparation to a 3D dataset

```
# Procedure a & b: Padding and Separate the data
look_back = 3
trainX, trainY = create_dataset(normalized_data, look_back)

# Procedure c: Reshape into 3D dataset
# [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], look_back, 1))
```

```
# Utility function
# convert an array of values into a input feature and target value
def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back):
                a = dataset[i:(i+look_back), 0]
                dataX.append(a)
                dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)
```

Suggested steps for LSTM coding (3)

- Model specification

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

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_dim=look_back))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Suggested steps for LSTM coding (4 & 5)

- Model training (tackle over-fit issue)

```
# Model training (without validation dataset)
model.fit(trainX, trainY, nb_epoch=100, batch_size=100)

# Model training (with validation dataset, prevent over-fit)
model.fit(trainX, trainY, nb_epoch=100, batch_size=100,
validation_data=(x_val, y_val) )
```

- Prediction

```
# Model prediction
testPredict = model.predict(testX)
```

Suggested steps for LSTM coding (6)

- Inverse transformation

```
# inverse transformation
testPredict = scaler.inverse_transform(testPredict)
```

Observations

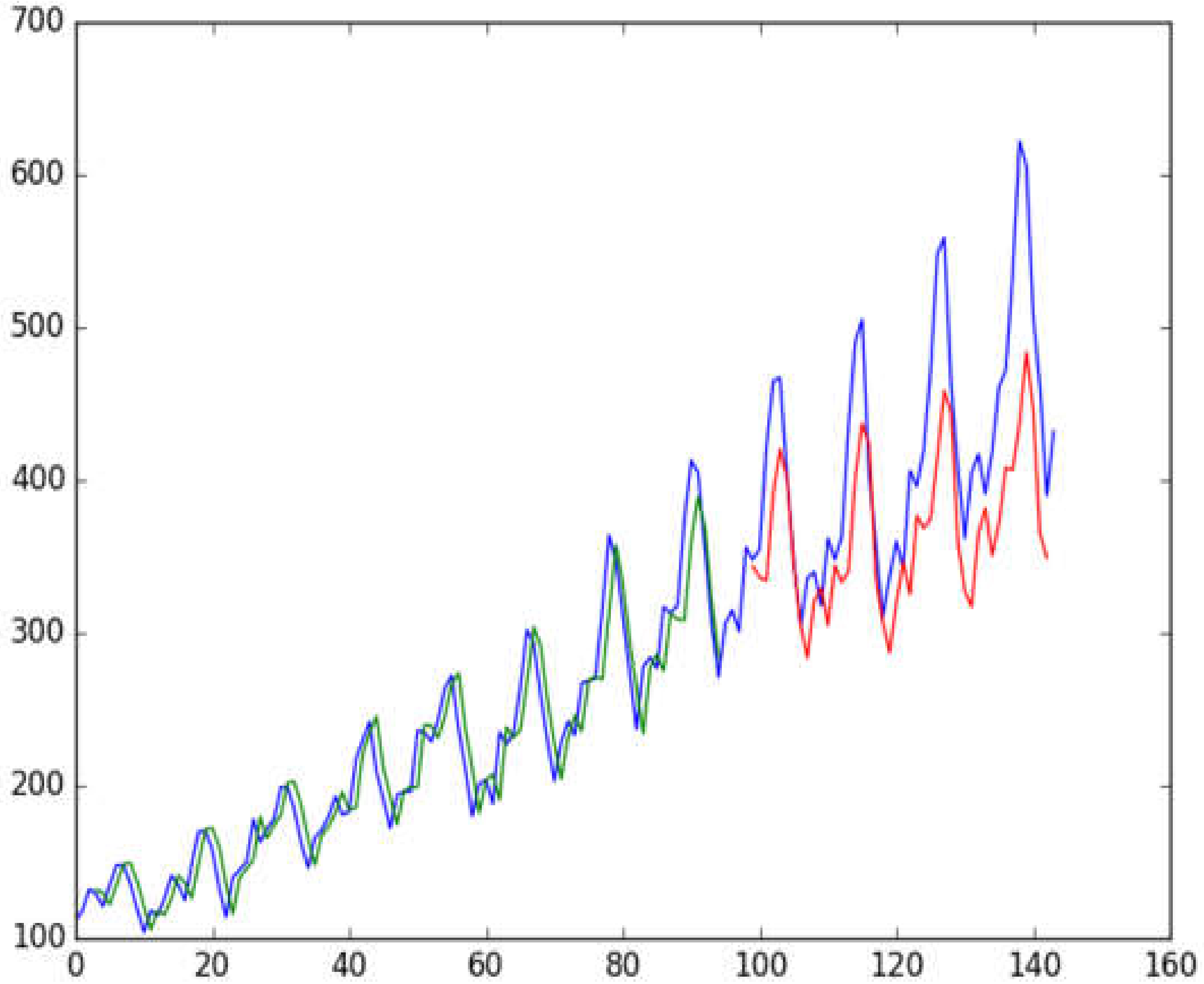
- How to frame the data in an appropriate way for sequence learning
  - Time-steps vs Features

- Normalization gives a better performance
  - Fewer epoch is needed for training
  - E.g. Epoch = >300 vs 100

```
import numpy
input_data=numpy.array([[1.0], [2.0], [3.0], [4],[5],[6],[7],[8],[9]])

#%% Step 1: normalize the dataset
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(-1, 1))
normalized_data = scaler.fit_transform(input_data)
plt.plot(normalized_data)

#%% Step 2: Data preparation to a 3D dataset
# Utility function
# convert an array of values into a input feature and target value
def data_preparation(input_data, model_input_length=1):
        dataX, target = [], []
        for i in range(len(input_data)-model_input_length):
                dataX.append(input_data[i:i+model_input_length, 0])
                target.append(input_data[i + model_input_length, 0])
        return numpy.array(dataX), numpy.array(target)

# Procedure a & b: Padding and Separate the data
model_input_length = 4  # the length of data used for modeling
trainX, trainY = data_preparation(normalized_data, model_input_length)

# Procedure c: Reshape into 3D dataset
# [samples, time steps, features]
trainX_3D_v1 = numpy.reshape(trainX, (trainX.shape[0], 1, model_input_length))  # misuse
trainX_3D_v2 = numpy.reshape(trainX, (trainX.shape[0], model_input_length, 1))  # correct

#%% Step 3: Model specification
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM

# create and fit the LSTM network
# Version 1: misuse
model_v1 = Sequential()
model_v1.add(LSTM(4, input_dim=look_back))
model_v1.add(Dense(1))
model_v1.compile(loss='mean_squared_error', optimizer='sgd')

# Version 2: correct usage
model_v2 = Sequential()
model_v2.add(LSTM(4, input_dim=1))
model_v2.add(Dense(1))
model_v2.compile(loss='mean_squared_error', optimizer='sgd')

#%% Step 4: Model training
# Model training (without validation dataset)
model_v1.fit(trainX_3D_v1, trainY, nb_epoch=200, batch_size=100, verbose=2)
model_v2.fit(trainX_3D_v2, trainY, nb_epoch=200, batch_size=100, verbose=2)

#%% Step 5: Model prediction
testX=numpy.array([[3.0], [4.0], [5.0], [6.0]])
# pay special attention on it....
normalized_testX = scaler.transform(testX)  # do not use fit_transform

testX_3D_v1=numpy.reshape(normalized_testX, (1, 1, look_back))
testPredict_v1 = model_v1.predict(testX_3D_v1)

testX_3D_v2=numpy.reshape(normalized_testX, (1, look_back, 1))
testPredict_v2 = model_v2.predict(testX_3D_v2)

#%% Step 6: inverse transformation
testPredict_final_v1 = scaler.inverse_transform(testPredict_v1)
testPredict_final_v2 = scaler.inverse_transform(testPredict_v2)

print('*** Final result: Version 1 ***')
print(testPredict_final_v1)
print('*** Final result: Version 2 ***')
print(testPredict_final_v2)
```

Example code of time series modeling using Keras (1)

```
# LSTM for international airline passengers problem with window regression framing
import numpy
import matplotlib.pyplot as plt
import pandas
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
	dataX, dataY = [], []
	for i in range(len(dataset)-look_back-1):
		a = dataset[i:(i+look_back), 0]
		dataX.append(a)
		dataY.append(dataset[i + look_back, 0])
	return numpy.array(dataX), numpy.array(dataY)
# fix random seed for reproducibility
numpy.random.seed(7)
# load the dataset
dataframe = pandas.read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# reshape into X=t and Y=t+1
look_back = 3
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_dim=look_back))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=100, batch_size=1, verbose=2)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



LSTM Trained on Window Method Formulation of Passenger Prediction Problem

- Ref: http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Example code of time series modeling using Keras (2)

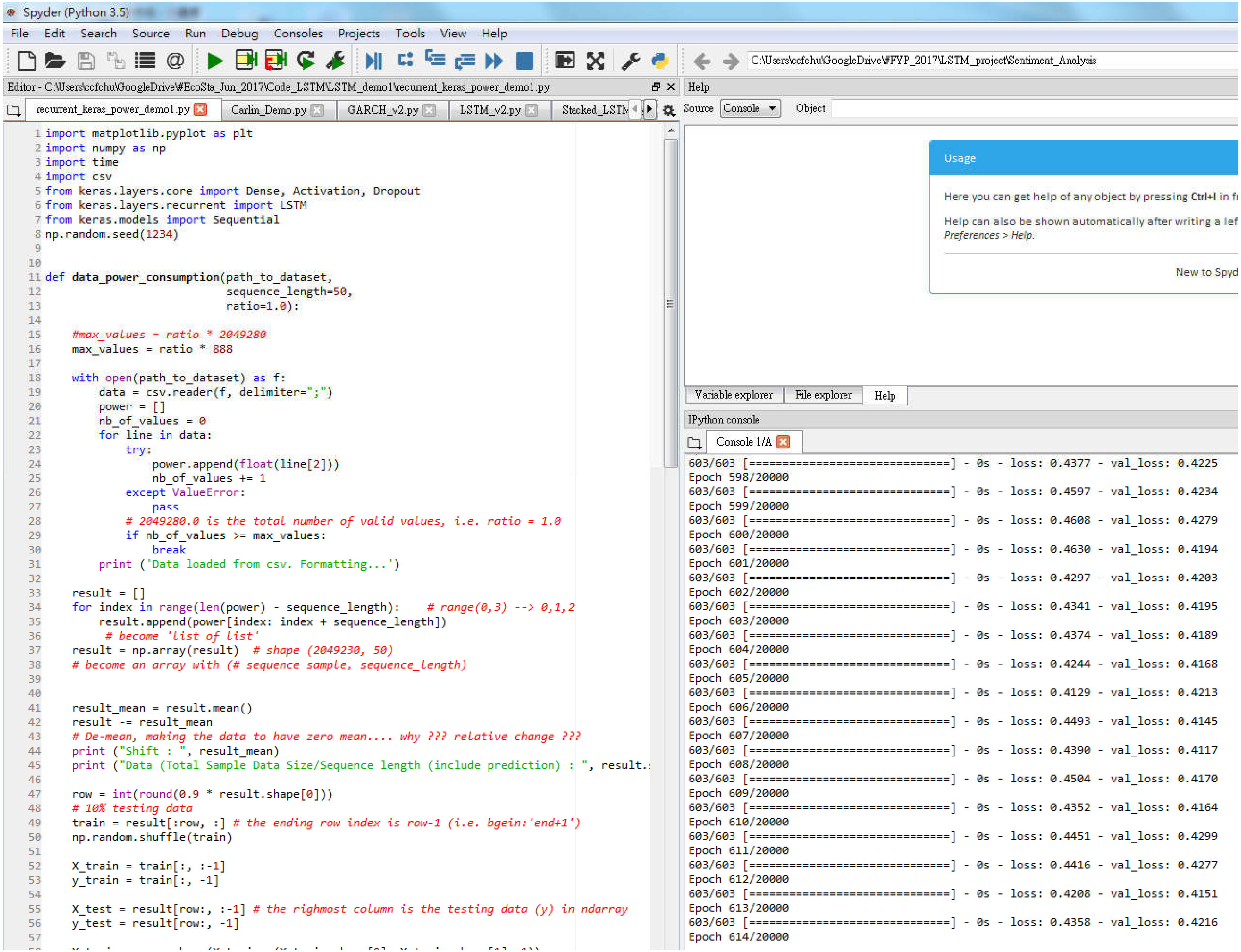
```
# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))

#%% Preparing training data
# prepare the dataset of input to output pairs encoded as integers
seq_length = 3
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
        seq_in = alphabet[i:i + seq_length]
        seq_out = alphabet[i + seq_length]
        dataX.append([char_to_int[char] for char in seq_in])
        dataY.append(char_to_int[seq_out])
        print (seq_in, '->', seq_out)

#%% We need to reshape the NumPy array into a format expected by the LSTM networks,
that is [samples, time steps, features].
# reshape X to be [samples, time steps, features]
X1 = numpy.reshape(dataX, (len(dataX), seq_length, 1))

# Once reshaped, we can then normalize the input integers to the range 0-to-1, the range of
the sigmoid activation functions used by the LSTM network.
# normalize
X1 = X1 / float(len(alphabet))

# Finally, we can think of this problem as a sequence classification task, where each of the 26
letters represents a different class.
# As such, we can convert the output (y) to a one hot encoding
# one hot encode the output variable
y1 = np_utils.to_categorical(dataY)
```

```
# %% create and fit the model
model1 = Sequential()
model1.add(LSTM(32, input_shape=(X1.shape[1], X1.shape[2])))
model1.add(Dense(y1.shape[1], activation='softmax'))
model1.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#numpy.random.seed(176)
model1.fit(X1, y1, nb_epoch=500, batch_size=1, verbose=2)

# After we fit the model we can evaluate and summarize the performance
# summarize performance of the model
scores = model1.evaluate(X1, y1, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))


# %% We can then re-run the training data through the network and generate
predictions,
# converting both the input and output pairs back into their original character format to
get a visual idea of how well the network learned the problem.

# demonstrate some model predictions
for pattern in dataX:
        x = numpy.reshape(pattern, (1, len(pattern), 1))
        x = x / float(len(alphabet))
        prediction = model1.predict(x, verbose=0)
        index = numpy.argmax(prediction)
        result = int_to_char[index]
        seq_in = [int_to_char[value] for value in pattern]
        print (seq_in, "->", result)
```

http://machinelearningmastery.com/understanding-stateful-lstm-recurrent-neural-networks-python-keras/

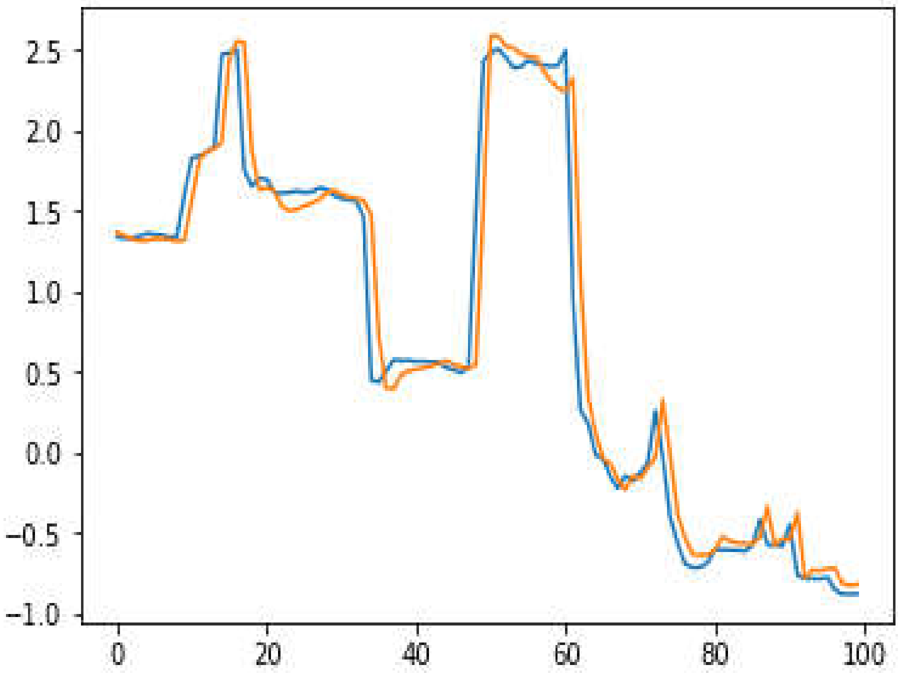
```
Epoch 500/500
0s - loss: 0.1935 - acc: 1.0000
Model Accuracy: 100.00%
['A', 'B', 'C'] -> D
['B', 'C', 'D'] -> E
['C', 'D', 'E'] -> F
['D', 'E', 'F'] -> G
['E', 'F', 'G'] -> H
['F', 'G', 'H'] -> I
['G', 'H', 'I'] -> J
['H', 'I', 'J'] -> K
['I', 'J', 'K'] -> L
['J', 'K', 'L'] -> M
['K', 'L', 'M'] -> N
['L', 'M', 'N'] -> O
['M', 'N', 'O'] -> P
['N', 'O', 'P'] -> Q
['O', 'P', 'Q'] -> R
['P', 'Q', 'R'] -> S
```

Example code of time series modeling using Keras (3)



https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent

```
In [12]: runfile('D:/data/Desktop/LSTM_demo1/recurrent_keras_power.py', wdir='D:/data/Desktop/LSTM_demo1')
Train on 876024 samples, validate on 46107 samples
Epoch 1/1
876024/876024 [==============================] - 1379s - loss: 0.0991 - val_loss: 0.0783
```



Suggested setting for LSTM Hyperparameter Tuning

- For LSTMs, use the **softsign** activation function over tanh (it's faster and less prone to saturation (vanishing gradient) (~0 gradients)).
- https://deeplearning4j.org/lstm.html

x	Softsign - level 1	Softsign - level 2	Softsign - level 3	Softsign - level 4	Slope	x	Tanh - level 1	Tanh - level 2	Tanh - level 3	Tanh - level 4	Slope
36	0.972973	0.493151	0.330275	0.248276	4.891E-05	36	1	0.761594	0.642015	0.56627	0
35	0.972222	0.492958	0.330189	0.248227	5.177E-05	35	1	0.761594	0.642015	0.56627	0
34	0.971429	0.492754	0.330097	0.248175	5.488E-05	34	1	0.761594	0.642015	0.56627	0
33	0.970588	0.492537	0.33	0.24812	5.829E-05	33	1	0.761594	0.642015	0.56627	0
32	0.969697	0.492308	0.329897	0.248062	6.202E-05	32	1	0.761594	0.642015	0.56627	0
31	0.96875	0.492063	0.329787	0.248	6.612E-05	31	1	0.761594	0.642015	0.56627	0
30	0.967742	0.491803	0.32967	0.247934	7.064E-05	30	1	0.761594	0.642015	0.56627	0
29	0.966667	0.491525	0.329545	0.247863	7.564E-05	29	1	0.761594	0.642015	0.56627	0
28	0.965517	0.491228	0.329412	0.247788	8.119E-05	28	1	0.761594	0.642015	0.56627	0
27	0.964286	0.490909	0.329268	0.247706	8.737E-05	27	1	0.761594	0.642015	0.56627	0
26	0.962963	0.490566	0.329114	0.247619	9.43E-05	26	1	0.761594	0.642015	0.56627	0
25	0.961538	0.490196	0.328947	0.247525	0.0001021	25	1	0.761594	0.642015	0.56627	0
24	0.96	0.489796	0.328767	0.247423	0.0001109	24	1	0.761594	0.642015	0.56627	0
23	0.958333	0.489362	0.328571	0.247312	0.0001208	23	1	0.761594	0.642015	0.56627	0
22	0.956522	0.488889	0.328358	0.247191	0.0001322	22	1	0.761594	0.642015	0.56627	0
21	0.954545	0.488372	0.328125	0.247059	0.0001452	21	1	0.761594	0.642015	0.56627	0
20	0.952381	0.487805	0.327869	0.246914	0.0001603	20	1	0.761594	0.642015	0.56627	0
19	0.95	0.487179	0.327586	0.246753	0.0001779	19	1	0.761594	0.642015	0.56627	0
18	0.947368	0.486486	0.327273	0.246575	0.0001985	18	1	0.761594	0.642015	0.56627	0
17	0.944444	0.485714	0.326923	0.246377	0.000223	17	1	0.761594	0.642015	0.56627	3.553E-15
16	0.941176	0.484848	0.326531	0.246154	0.0002522	16	1	0.761594	0.642015	0.56627	2.72E-14
15	0.9375	0.483871	0.326087	0.245902	0.0002876	15	1	0.761594	0.642015	0.56627	2.004E-13
14	0.933333	0.482759	0.325581	0.245614	0.000331	14	1	0.761594	0.642015	0.56627	1.482E-12

Bollerslev, Patton and Quaedvlieg (2016)

- **Improved version** for time series modeling of realized variance
- Improved **Heterogeneous Autoregressive regression (HAR)**

$$RV_{t+1d}^{d} = c + \beta^{d} RV_{t}^{d} + \beta^{w} RV_{t}^{w} + \beta^{m} RV_{t}^{m} + \omega_{t+1d}$$ **Typical HAR : Daily, Weekly, Monthly**

$$RV_t = \beta_0 + \underbrace{\left(\beta_1 + \beta_{1Q} RQ_{t-1}^{1/2}\right)}_{\beta_{1,t}} RV_{t-1} + \beta_2 RV_{t-1|t-5} + \beta_3 RV_{t-1|t-22} + u_t$$

$$RV_t \equiv \sum_{i=1}^{M} r_{t,i}^2 \quad RQ_t \equiv \frac{M}{3} \sum_{i=1}^{M} r_{t,i}^4 \quad RV_{t-j|t-h} = \frac{1}{h} \sum_{i=j}^{h} RV_{t-i}$$

Work in progress

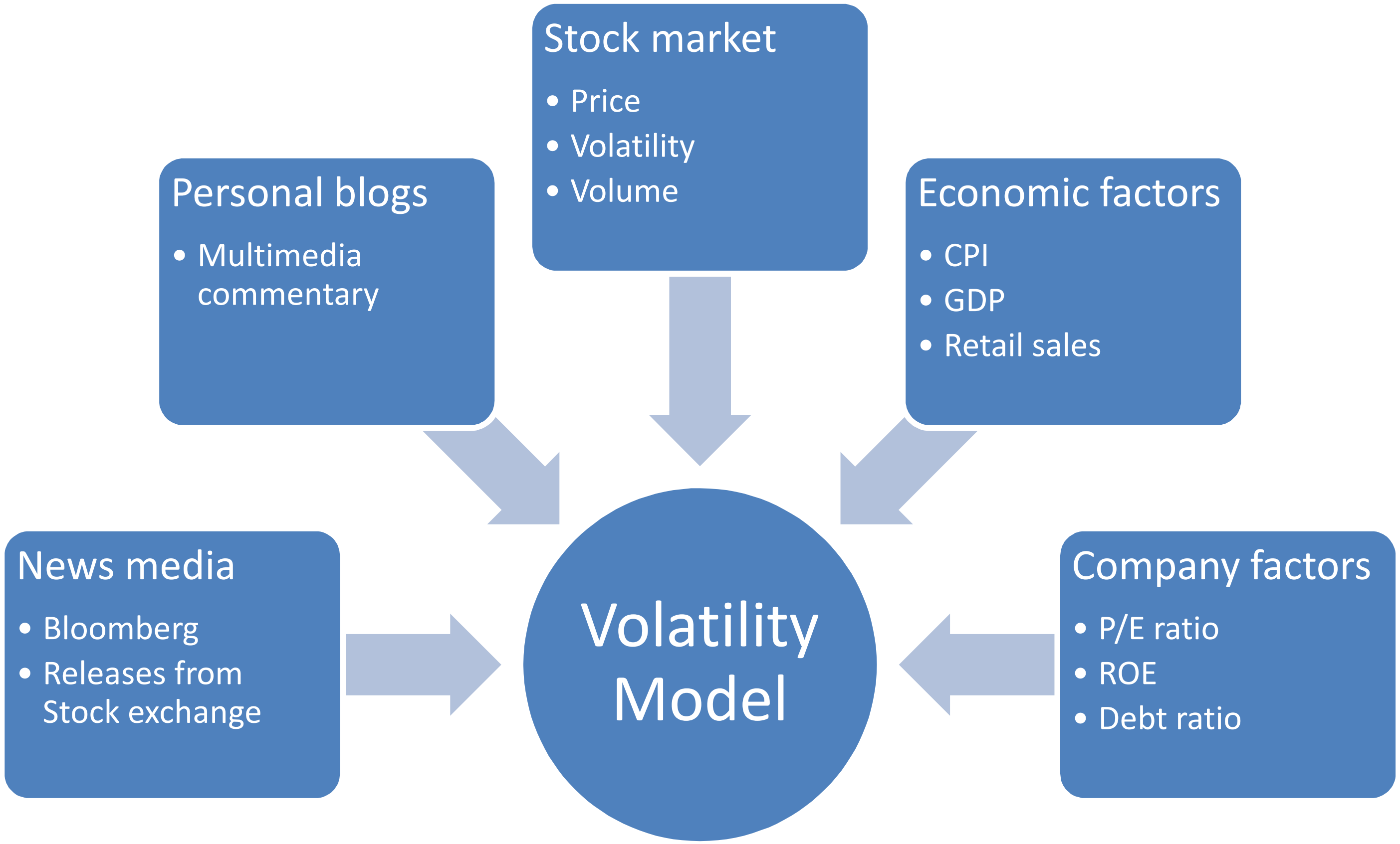
Long Short Term Memory network (LSTM)

- Adaptive forget gate, throw away information
- Keep information with time gaps of unknown/different size(s)

Investigation:

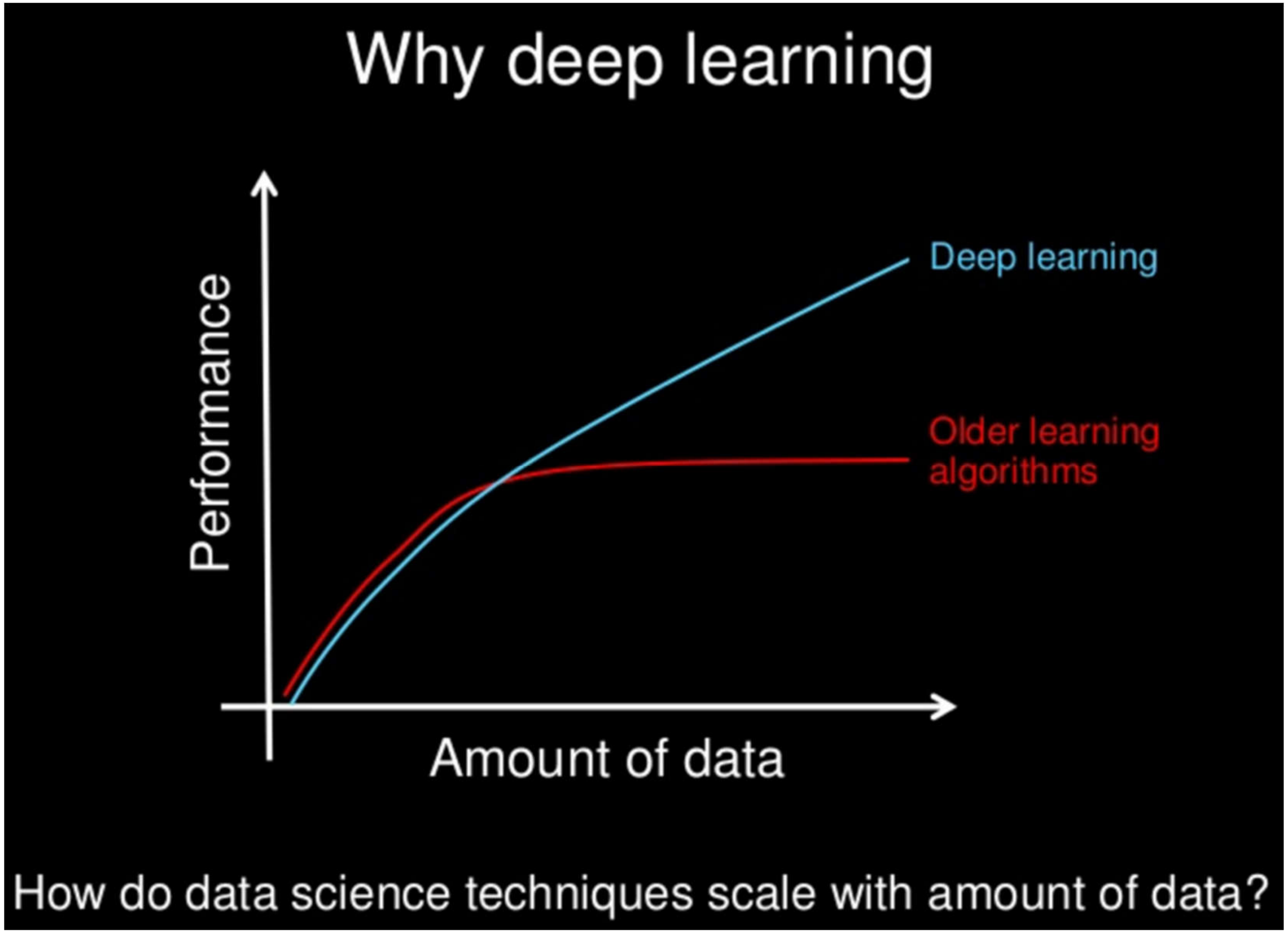
- Possible to extract features from different time horizons ?
  - Daily, Weekly, Monthly, Intraday
- Model structure ?
  - Number of layers ? Activation functions
- How to prevent over-fitting ?
  - Types of loss function
- What types of information can be used ?
  - Numerical, News, Comment from Social network

Make use of different types of information ?



Machine Learning techniques (more flexible)
Time series approach (more rigid)

If everything goes right …



The picture is extracted from: http://machinelearningmastery.com/what-is-deep-learning/
Why Deep Learning? (Slide by Andrew Ng, Stanford University)

Thank you for your kind attention.

Hope you find this presentation interesting.

Reference

- A Beginner's Guide to Recurrent Networks and LSTMs - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM
- https://deeplearning4j.org/lstm.html
- Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras (by Jason Brownlee)
- http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
- Understanding Stateful LSTM Recurrent Neural Networks in Python with Keras (by Jason Brownlee)
- http://machinelearningmastery.com/understanding-stateful-lstm-recurrent-neural-networks-python-keras/
- Keras recurrent tutorial
- https://github.com/Vict0rSch/deep_learning/tree/master/keras/recurrent
- Deep learning Wikipedia
- https://en.wikipedia.org/wiki/Deep_learning
- What Is The Difference Between Deep Learning, Machine Learning and AI?
- https://www.forbes.com/sites/bernardmarr/2016/12/08/what-is-the-difference-between-deep-learning-machine-learning-and-ai/#496301bc26cf
- What is Deep Learning? (by Jason Brownlee)
- http://machinelearningmastery.com/what-is-deep-learning/
- Do you recommend using Theano or Tensor Flow as Keras' backend? - Quora
- https://www.quora.com/Do-you-recommend-using-Theano-or-Tensor-Flow-as-Keras-backend
- Backend - Keras Documentation
- https://keras.io/backend/
- Ill-Conditioning in Neural Networks
- ftp://ftp.sas.com/pub/neural/illcond/illcond.html
- Understanding LSTM Networks
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Getting started with the Keras Sequential model
- https://keras.io/getting-started/sequential-model-guide/
- Tim Bollerslev, Andrew J. Patton, Rogier Quaedvlieg (2016) Exploiting the errors: A simple approach for improved volatility forecasting, Journal of Econometrics, Volume 192, Issue 1, 2016, Pages 1-18