Machine Learning

Time Series Forecasting
LSTM

CE 5331 Machine Learning for Civil Engineers

Venki Uddameri,
Ph.D. , P.E.

Problem Statement

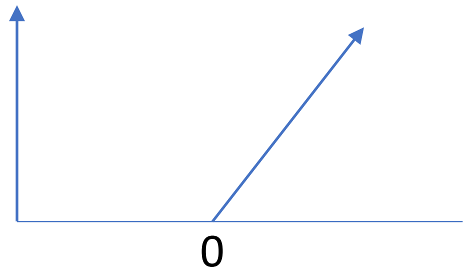
- Fit the Monthly Streamflow time-series data using
  - Precipitation and Temperature as exogeneous variables
  - Lag 1 of precipitation and temperature as exogeneous variable
  - Lags of streamflow as endogenous variable
- Data provided to you in StreamflowMonthly.csv

Step 1: Data Processing

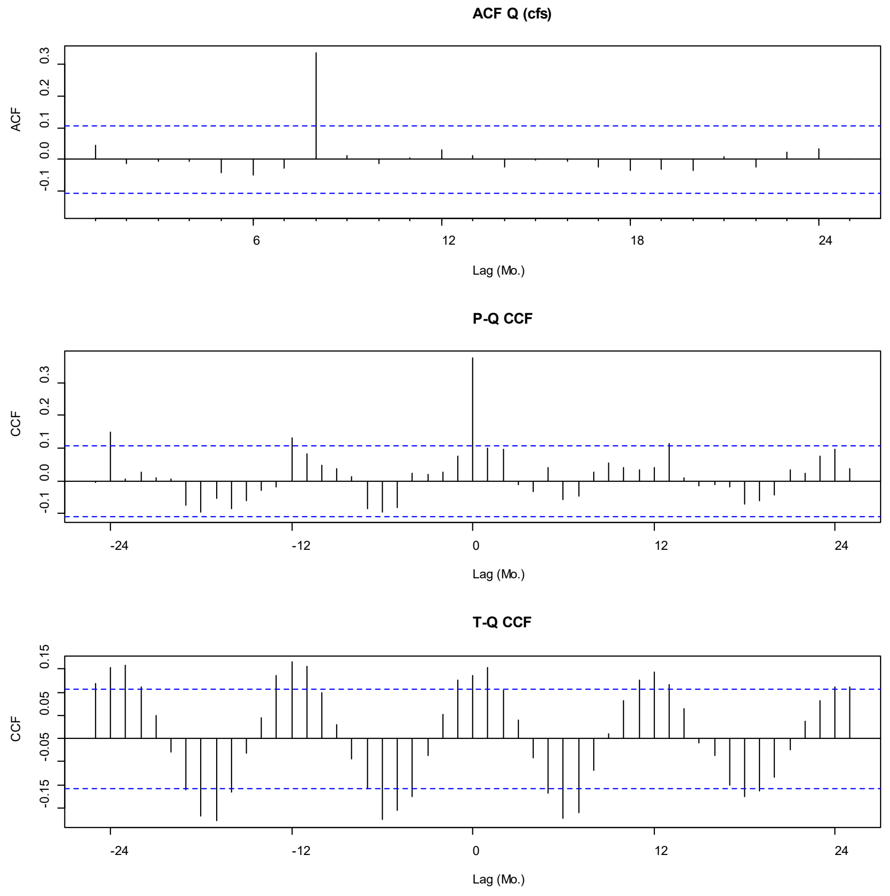
- We need to add lag terms
  - While this can be done in Python using Pandas it is perhaps easier to do this outside in Excel or R
  - Check how the ACF and CCF Look
- We need to add identify important variables
  - We can use a variety of techniques
  - We shall use Random Forest based Importance
    - Decrease in mean accuracy
    - Can implement using package “Fselector” in R

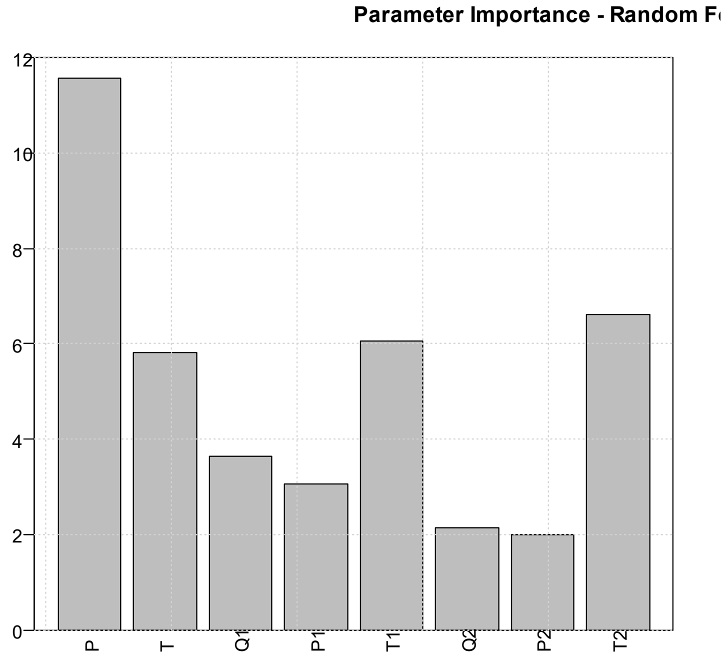
```
# Script to Identify important variables
# Venki Uddameri, Ph.D. P.E. 4/28/2020
# Load libraries
library(FSelector)
library(forecast)

# Set working Directory and read data
setwd('D:\\Dropbox\\000CE5333Machine Learning\\Week10TimeSeries\\Code')
a <- read.csv('MonthlyStreamFlowlag2.csv')
parm <- a[,4:12]

# Plot ACF and CCF
parm.ts <- ts(parm,start= c(1988,5),frequency=12)

Q.ts <- parm.ts[,1]
P.ts <- parm.ts[,2]
T.ts <- parm.ts[,3]

par(mfrow=c(3,1))
Acf(Q.ts,main='ACF Q (cfs)',xlab='Lag (Mo.)')
Ccf(P.ts,Q.ts, main='P-Q CCF',xlab='Lag (Mo.)')
Ccf(T.ts,Q.ts, main='T-Q CCF',xlab='Lag (Mo.)')

# Feature Selection using correlation and entropy method
parm.cfs <- cfs(Q~.,data=parm)

# Using Random Forect Method
parm.rf <- random.forest.importance(Q~.,parm, importance.type = 1)
names = rownames(parm.rf)
barplot(as.vector(unlist(parm.rf)),names.arg=names,las=2,ylim=c(0,12))
box()
grid()
title('Parameter Importance - Random Forest')
```

Step 1: EDA



- Lag 8 of Q seems prominent
  - No real physical explanation
  - Late Summer – Spring Flows
- Q shows cyclicity with temperature
- Precipitation lags not as significant
  - Except for zero lag





Retain Precipitation, Temperature and its 2 lags (endogenous variables);
May be add lag 1 Q (exogenous)

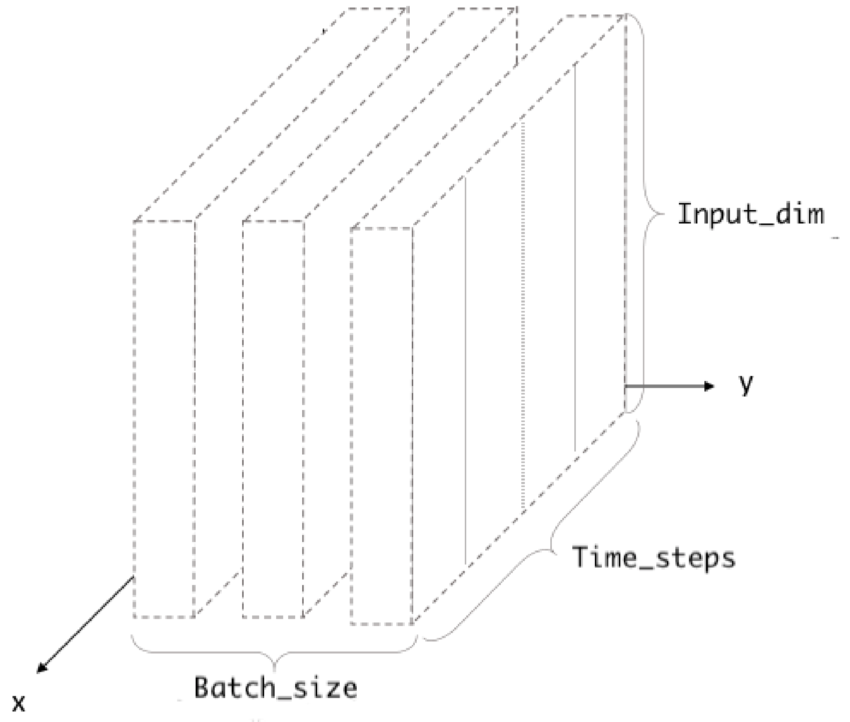
Step 2: Data Processing for LSTM

- You can read the dataset into python using pandas
- You can extract the variables as usual
    - Input
    - Output
- Training and Testing splits
    - Same as before (70 – 30; 80 – 20, etc based on data)
    - Split is not randomized in Time Series
        - Use early part to split and later part to test
        - Stationarity issues
- Normalization of input and output data are critical for LSTMs
    - Normalization typically between 0 and 1
    - Need to revert back to the original for post-processing
    - Usually the only the training data min and max are used for normalization
        - Good to write you own function rather than use sklearn MinMaxScaler
        - Cannot handle non-stationarity well

```
# function to normalize
def minmaxscl(x,n):
    N = len(x)
    x_trn = np.array(x[0:n])
    xx = np.array(x)
    minx = np.min(x_trn)
    maxx = np.max(x_trn)
    x_tr = (xx-minx)/(maxx-minx)
    return(x_tr)

# function to make inv transformation
def invminmaxscl(xinv,x,n):
    N = len(x)
    x_trn = np.array(x[0:n])
    minx = np.min(x_trn)
    maxx = np.max(x_trn)
    dnum = maxx - minx
    xinv = np.array(xinv)
    Xp = xinv*dnum+minx
    return(Xp)
```

Step 3: Data Processing

- Getting Input and Output Data in proper format is critical for implementing LSTM in Keras
- Keras assumes your input data has 3 elements
    - **Samples**. One sequence is one sample. A batch is comprised of one or more samples.
    - **Time Steps**. One time step is one point of observation in the sample.
    - **Features**. One feature is one attribute observation at a time step.



Input Data is usually in the form of 1D (single attribute) and 2D (multiattribute) Arrays

Step 3: Data Processing Cont..

- Data needs to be reshaped to confirm to LSTM specifications
- Use reshape function in python to reshape 1D and 2D arrays

```
# Example of 1D input Array
import numpy as np
data = np.array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
data = data.reshape((10, 1, 1))
```

Features

Time-Steps

Samples

2 columns

```
data2D = array([[0.1,
0.2],
       [0.3, 0.4],
       [0.5, 0.6],
       [0.7, 0.8],
       [0.9, 1. ]])
```

5 Rows

```
import numpy as np
data2D = data.reshape((5, 1, 2))
print(data2D.shape)
```

Samples

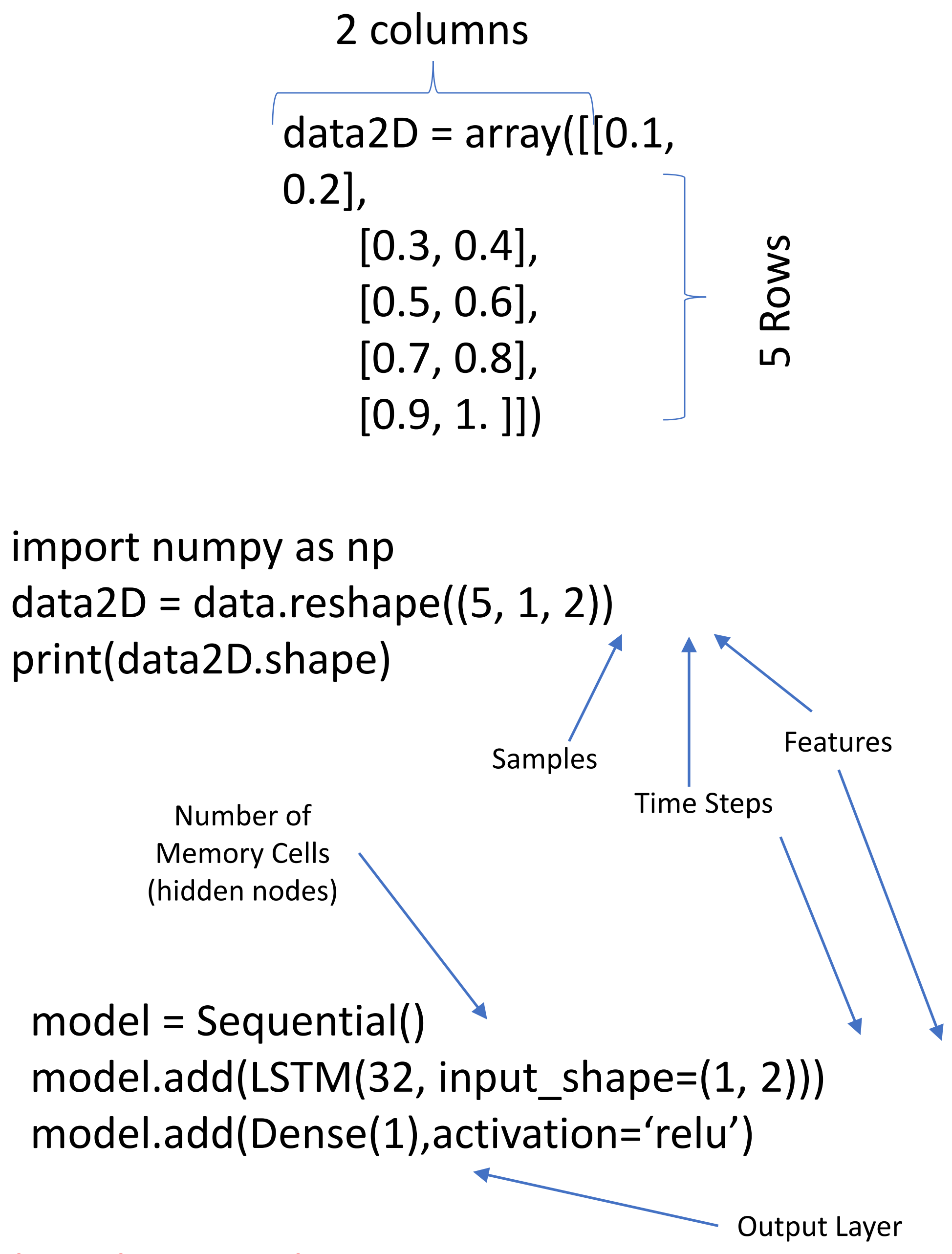
Time Steps

Features

Keras assumes there is at least one or more sample(s)

Specifying the LSTM Layer

- LSTM is a sequential model
  - Input → Memory Cells → Output
- The First layer connecting Input to Memory Cells
  - Need to specify the shape of the input (# of time-steps, # of Features)
    - The Number of samples is based on the batch size
    - Batch size is usually taken as 1 but need not be
      - Remember the smaller the batch the better



```
data2D = array([[0.1,
0.2],
          [0.3, 0.4],
          [0.5, 0.6],
          [0.7, 0.8],
          [0.9, 1. ]])
```

```
import numpy as np
data2D = data.reshape((5, 1, 2))
print(data2D.shape)
```

```
model = Sequential()
model.add(LSTM(32, input_shape=(1, 2)))
model.add(Dense(1),activation='relu')
```

The batch size should be a multiple of samples otherwise cannot be split properly

LSTM Model Specification

- Too many epochs overfitting
- Too few epochs underfitting
- LSTMs are prone to overfitting
  - Compare tes

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(100, input_shape=(1,4)))
model.add(Dense(1,activation='relu’))

# Compile Model
model.compile(loss='mean_squared_error', optimizer='adam’,)

#Fit the Model
model.fit(trainX, trainy, epochs=100, batch_size=1,
verbose=2,shuffle=False)
```

Python Code

```
# Import libraries
import os
import pandas as pd
import numpy as np
import tensorflow as tf
from keras.models import
Sequential
from keras.layers import Dense
from keras.layers import LSTM’
from datetime import datetime
from matplotlib import pyplot as plt
```

```
# function to normalize
def minmaxscl(x,n):
    N = len(x)
    x_trn = np.array(x[0:n])
    xx = np.array(x)
    minx = np.min(x_trn)
    maxx = np.max(x_trn)
    x_tr = (xx-minx)/(maxx-minx)
    return(x_tr)

# function to make inv transformation
def invminmaxscl(xinv,x,n):
    N = len(x)
    x_trn = np.array(x[0:n])
    minx = np.min(x_trn)
    maxx = np.max(x_trn)
    dnum = maxx - minx
    xinv = np.array(xinv)
    Xp = xinv*dnum+minx
    return(Xp)
```

```
# fix random seed for reproducibility
np.random.seed(7)

os.chdir('D:\\Dropbox\\000CE5333Machine Learning\\Week10TimeSeries\\Code')
a = pd.read_csv('MonthlyStreamflowlag2.csv', header=0, parse_dates=[0], index_col=0)

features = ['T','T1','T2','Q1'] # Input data based on RF
X = a[features] # Dataframe
Y = a['Q'] # Output
Q = a['Q']
# Training and Testing Splits
N = int(len(X))
splt = 0.8
n = int(N*splt)
na = N + 1
```

```
#scale the data
Y_scl = minmaxscl(Y,n) # use with scalar
T_scl = minmaxscl(X['T'],n)
T1_scl = minmaxscl(X['T1'],n)
T2_scl = minmaxscl(X['T2'],n)
Q1_scl = minmaxscl(X['Q1'],n)
X_scl = pd.DataFrame({'T':T_scl,'T1':T1_scl,'T2':T2_scl,'Q1':Q1_scl})
X_scl_np = np.array(X_scl)
Y_scl_np = np.array(Y_scl)

# split into train and test sets
trainX, trainy = X_scl_np[0:n,], Y_scl_np[0:n]
testX, testy = X_scl_np[n:na,], Y_scl_np[n:na]
```

Python Code Cont..

```
# reshape input to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(100, input_shape=(1,4)))
model.add(Dense(1,activation='relu'))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainy, epochs=100, batch_size=1, verbose=2,shuffle=False)
```

Keras Model Fitting usually shuffles the data between epochs
So set it to false to preserve time-series structure

```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# invert predictions
Qp_train = invminmaxscl(trainPredict,Y,n)
Qp_test = invminmaxscl(testPredict,Y,n)
Qp =  np.concatenate([Qp_train,Qp_test], axis=0)
Qp = Qp.ravel()
date = np.asarray(Y.index)
Qp = pd.Series(Qp, index =date)
```

```
# Make a Basic Plot
plt.plot(Q)
plt.plot(Qp)
plt.xlabel('Year')
plt.ylabel('Q (cfs)')
plt.grid()

# Write to a csv file for futther processing
df = pd.concat([Q, Qp], axis=1)
df.to_csv('results.csv')
```

Other Issues

- How to handle batch data
  - Stateless mode

    Earlier batch has no effect on the present batch
  - Stateful mode
    - Earlier batch fitting has an effect
    - Useful for highly correlated data
- LSTMs are prone to overfitting
  - Dropout
    - Input Dropout, Recurrent layer drop out
  - EarlyStopping
    - A call back during fitting

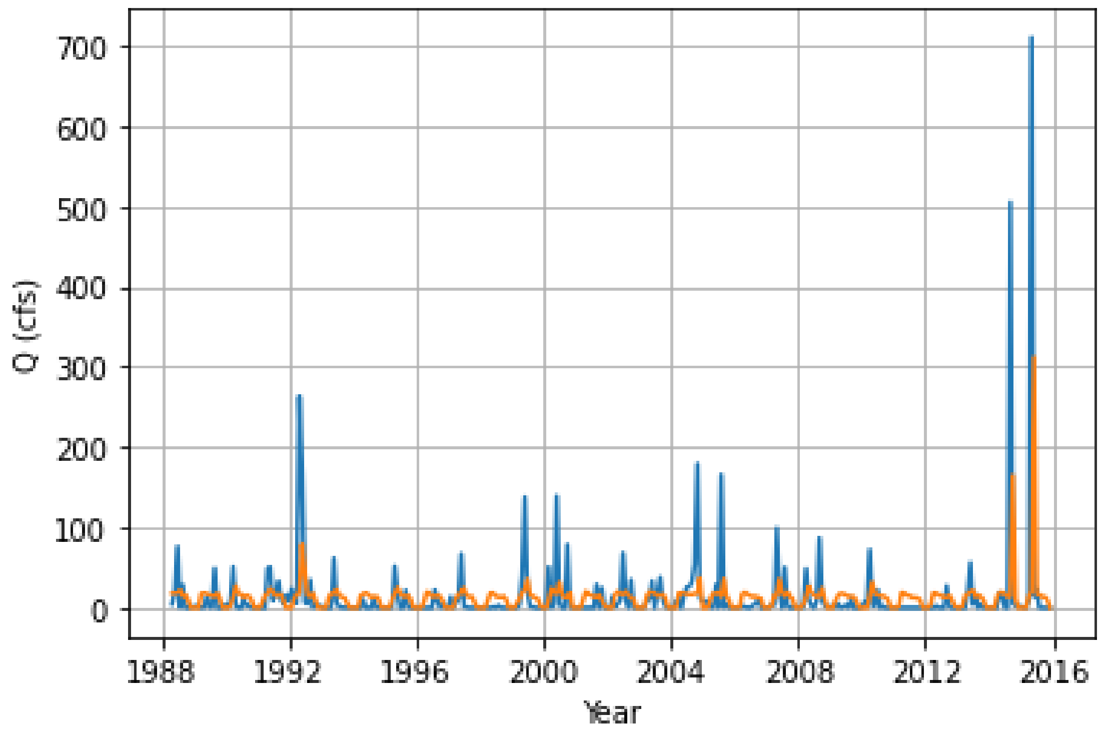
Noting beats visual comparisons

Goodness of Fit measures can be used once the models are fit

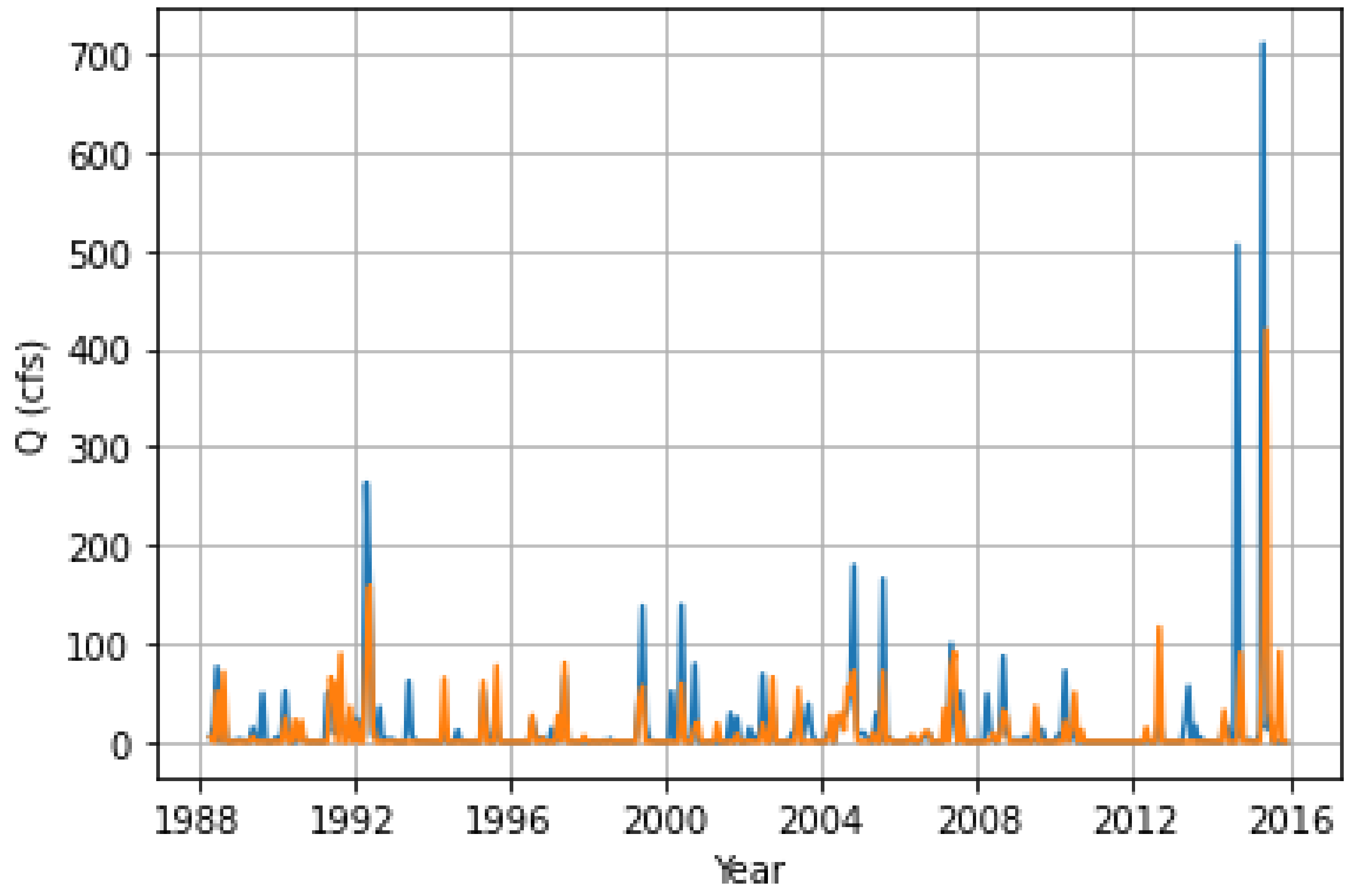
Need to retransform the data for comparisons

Results

- Model Captures the trend
- Not very good at capturing flashiness
  - Problem with most models with this type of data
  - Is it over a threshold to make a decision?
- ReLU is a good activation function for the linear layer
  - Model may predict negative flows which are physically unrealistic
- Overfitting does not appear to be a problem
- Model was not corrected for non-stationarity, but appears to still capture the trends



Results with P



Other Techniques

- Data can be taken in R for further analysis
  - Analyze predicted and observed time-series
- A variety of Goodness of Fit (GOF) methods can be used for assessing predicted and observed time series
  - HydroGOF package in R

```
# Comparing two time series using hydroGOF package
# Venki Uddameri
library(hydroGOF)
# Set working Directory
setwd('D:\\Dropbox\\000CE5333Machine Learning\\Week10TimeSeries\\Code')

# read data
a = read.csv('results.csv')
gof(a$Q,a$Qp)
KGE(a$Q,a$Qp,method='2012',out.type='full')
```

Results

```
ME        1.47
MAE       18.49
MSE     3194.73
RMSE      56.52
NRMSE %  273.00
PBIAS %   11.60
RSR        2.73
rSD        2.64
NSE       -6.48
mNSE      -1.21
rNSE        NaN
d          0.13
md         0.33
rd          NaN
cp        -3.42
r          0.09
R2         0.01
bR2        0.00
KGE       -0.88
VE        -0.46
```

```
$KGE.value
[1] -0.6410631

$KGE.elements
         r       Beta      Gamma
0.09126215 1.11649961 2.36151071
```

You should Know

- How to identify inputs for an LSTM model
  - Model selection using R packages
- How to set up a ‘vanilla’ LSTM model in Python
  - How to read the data
  - How to reshape the input
  - Issues to consider when setting up LSTM model
    - Number of cells
    - How to handle the output cell
      - Use of ReLU, Sigmoid, softmax and other activation functions
    - How to handle batches
      - Stateless versus state mode
    - Shuffling between the epochs
      - Should not be done
    - How to export results to a csv for further analysis
      - Use of R and KGE