In [1]:
```
# Based on Recurrent Neural Networks
%matplotlib inline

from __future__ import print_function

import os
os.environ["THEANO_FLAGS"] = "mode=FAST_RUN,device=gpu,floatX=float32"


import numpy as np
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

plt.style.use('ggplot')
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 960 (CNMeM is disabled, cuDNN not available)
```

In [2]:
```
# fix random seed for reproducibility
np.random.seed(10)
```

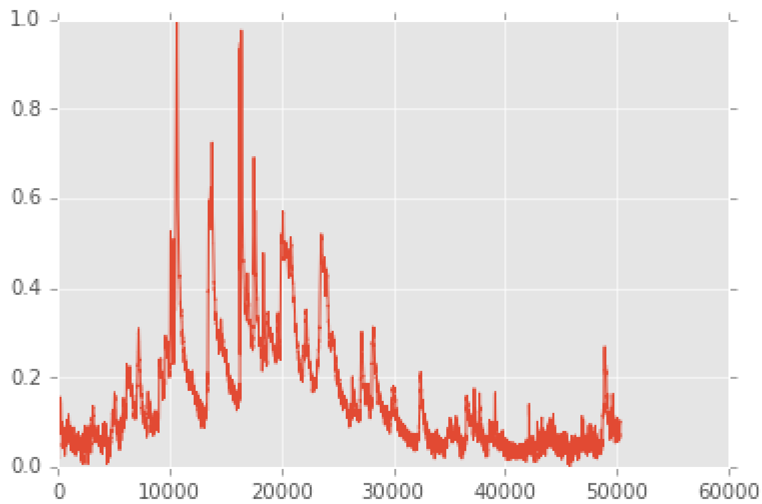
In [3]:
```
#Use the flood_data.csv dataset
dataframe = pandas.read_csv('dataset/flood_train.csv', usecols=[1], engine='pytho
dataset   = dataframe.values
dataset   = dataset.astype('float32')
dataframe.head()
```

Out[3]:

	waterlevel
0	0.27
1	0.26
2	0.27
3	0.28
4	0.28

In [18]:
```
plt.plot(dataset)
```

Out[18]:
```
[<matplotlib.lines.Line2D at 0x7fbfbccf5210>]
```



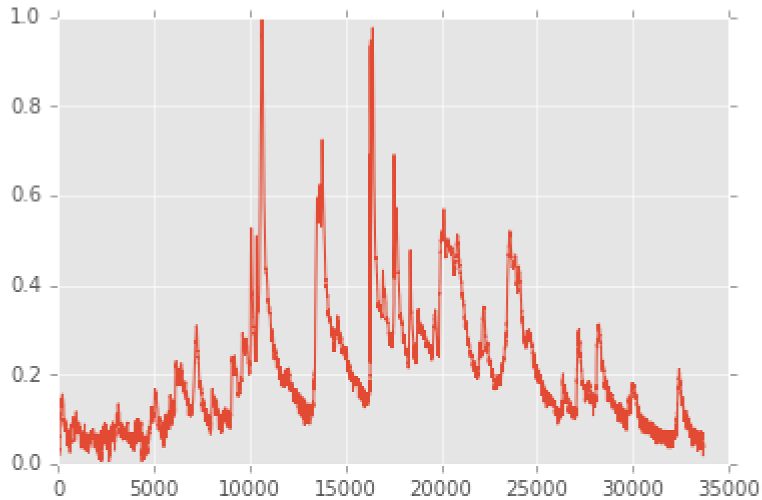
In [5]:
```
# normalize the dataset
scaler  = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

In [6]:
```
# 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),:]
print(len(train), len(test))
```

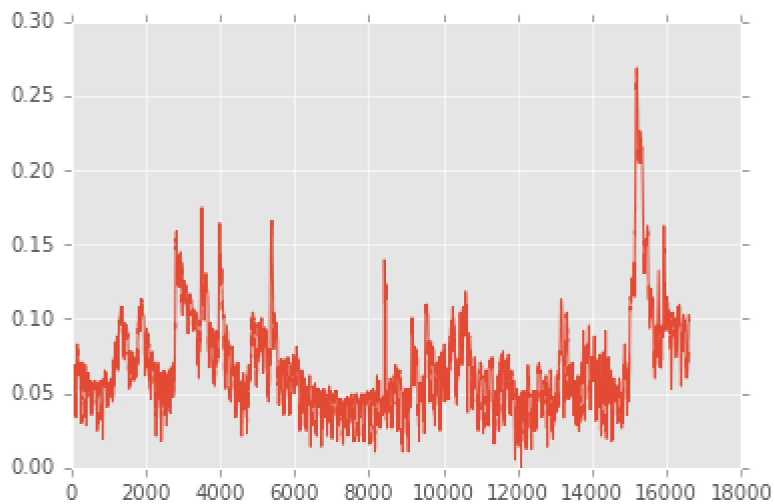
```
33710 16604
```

In [7]:
```
print('---------------- TRAINING DATA ----------------')
plt.plot(train)
plt.show()
print('---------------- TEST DATA ----------------')
plt.plot(test)
plt.show()
```

```
---------------- TRAINING DATA ----------------
```




```
---------------- TEST DATA ----------------
```



In [8]:
```
# This function creates a sliding window of the dataset.
def create_dataset(dataset, sliding_window=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-sliding_window-1):
        a = dataset[i:(i+sliding_window), 0]
        dataX.append(a)
        dataY.append(dataset[i + sliding_window, 0])
    return np.array(dataX), np.array(dataY)
```

In [9]:
```
# use a n-10 sliding window equivalent to 2.5 hours of historical data
slide_window   = 10
trainX, trainY = create_dataset(train, slide_window)
testX, testY   = create_dataset(test, slide_window)
```

In [10]:
```
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX  = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

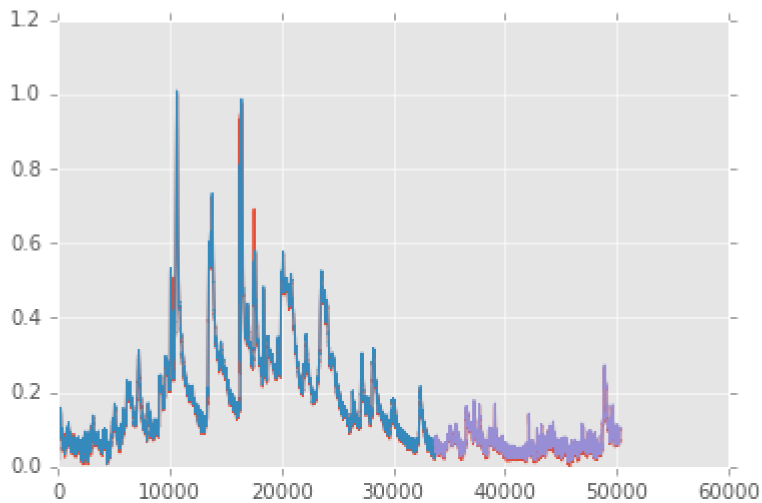
In [11]:
```
#Setup the LSTM

model = Sequential()
model.add(LSTM(4, input_dim=slide_window))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, nb_epoch=50, batch_size=1, verbose=2)
```

```
Epoch 1/70
104s - loss: 2.7772e-04
Epoch 2/70
98s - loss: 1.1746e-04
Epoch 3/70
98s - loss: 1.0950e-04
Epoch 4/70
99s - loss: 1.0798e-04
Epoch 5/70
99s - loss: 1.0540e-04
Epoch 6/70
98s - loss: 1.0358e-04
Epoch 7/70
98s - loss: 1.0405e-04
Epoch 8/70
98s - loss: 1.0088e-04
Epoch 9/70
98s - loss: 9.6544e-05
Epoch 10/70
```

In [12]:
```
# Print out the evaluation for both the
trainScore = model.evaluate(trainX, trainY, verbose=0)
trainScore = math.sqrt(trainScore)
trainScore = scaler.inverse_transform(np.array([[trainScore]]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = model.evaluate(testX, testY, verbose=0)
testScore = math.sqrt(testScore)
testScore = scaler.inverse_transform(np.array([[testScore]]))
print('Test Score: %.2f RMSE' % (testScore))
```

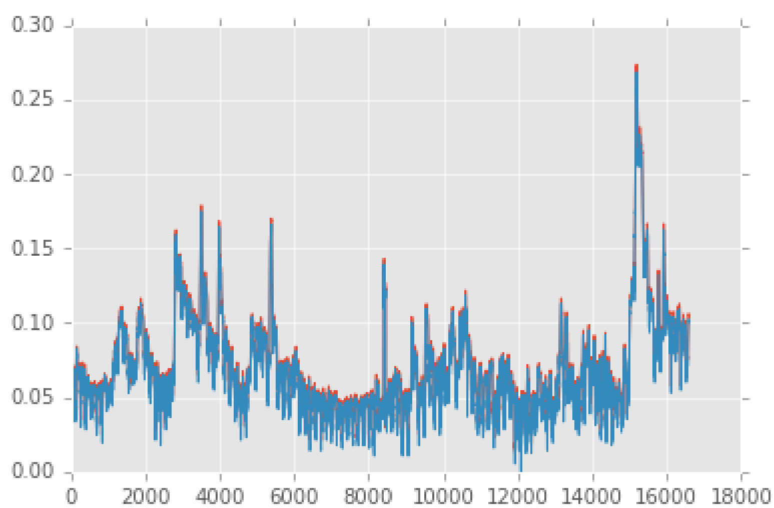
```
Train Score: 0.06 RMSE
Test Score: 0.03 RMSE
```

In [13]:
```
trainPredict = model.predict(trainX)
testPredict  = model.predict(testX)

# shift train predictions for plotting
trainPredictPlot       = np.empty_like(dataset)
trainPredictPlot[:, :] = np.nan
trainPredictPlot[slide_window:len(trainPredict)+slide_window, :] = trainPredict

# shift test predictions for plotting
testPredictPlot       = np.empty_like(dataset)
testPredictPlot[:, :] = np.nan
testPredictPlot[len(trainPredict)+(slide_window*2)+1:len(dataset)-1, :] = testPre

# plot baseline and predictions
plt.plot(dataset)
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```



In [16]:
```
plt.plot(testPredict)
plt.plot(testY)
```

Out[16]: [<matplotlib.lines.Line2D at 0x7fbfbd0a6910>]



In [20]:
```
# Test the network on an unseen data
unseen = pandas.read_csv('dataset/flood_test.csv',sep=',')
```

In [21]:
```
unseen.head()
```

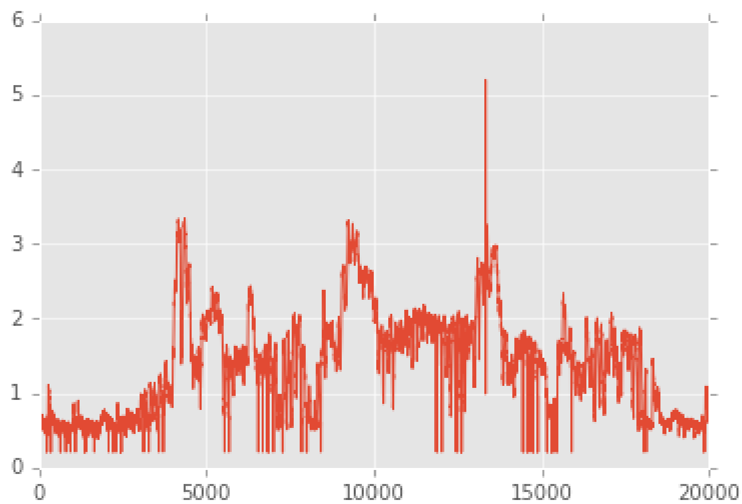
Out[21]:

	datetime	rainfall	waterlevel
0	1/1/2013 0:10	0.0	0.21
1	1/1/2013 0:21	0.0	0.21
2	1/1/2013 0:30	0.0	0.40
3	1/1/2013 1:30	0.0	0.49
4	1/1/2013 1:40	0.0	0.59

In [22]:
```
unseen_test = unseen['waterlevel'].values
```

In [24]:
```
plt.plot(unseen_test[0:20000])
```

Out[24]: [<matplotlib.lines.Line2D at 0x7fbfbc57b350>]



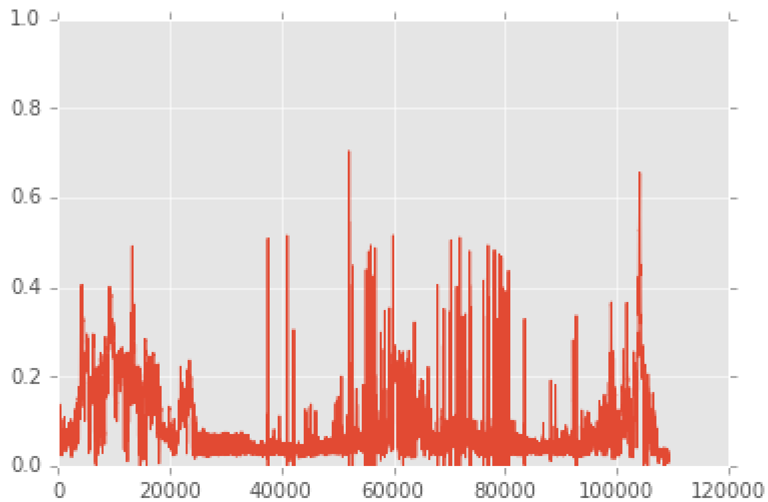
In [84]:
```
unseen_clean = []
for i in unseen_test:
    unseen_clean.append([i])
unseen_clean = np.asarray(unseen_clean).astype('float32')
unseen_clean = scaler.fit_transform(unseen_clean)
```

In [85]:
```
features,labels = create_dataset(unseen_clean, slide_window)
features        = np.reshape(features, (109186,1, 10))
```

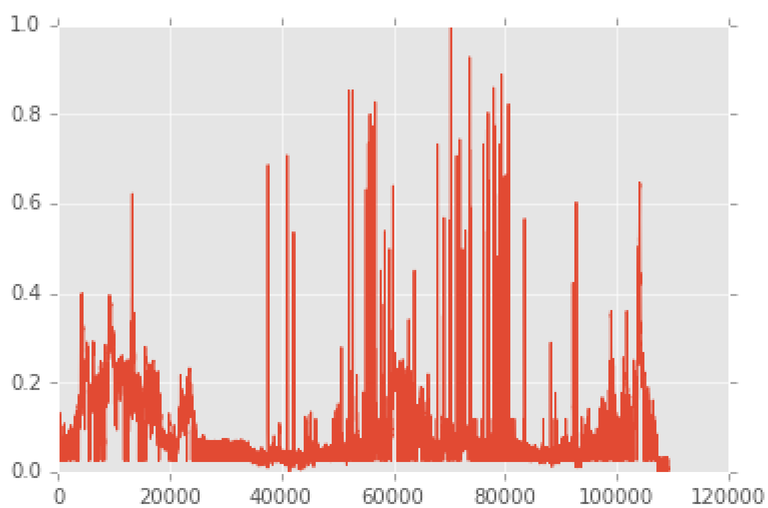
In [86]:
```
unseen_results = model.predict(features)
```

In [115]:
```
plt.gca().set_ylim(bottom=0)
plt.gca().set_ylim(top=1)

print('-------------------- Predicted --------------------')
plt.plot(unseen_results)
plt.show()


print('-------------------- Ground Truth --------------------')
plt.plot(labels)
plt.show()
```

```
-------------------- Predicted --------------------
```




```
-------------------- Ground Truth --------------------
```



In [112]:
```
# Check the root mean squared error for the new test set

testScore = model.evaluate(features, labels, verbose=0)
testScore = math.sqrt(testScore)
testScore = scaler.inverse_transform(np.array([[testScore]]))
print('Test Score: %.2f RMSE' % (testScore))
```

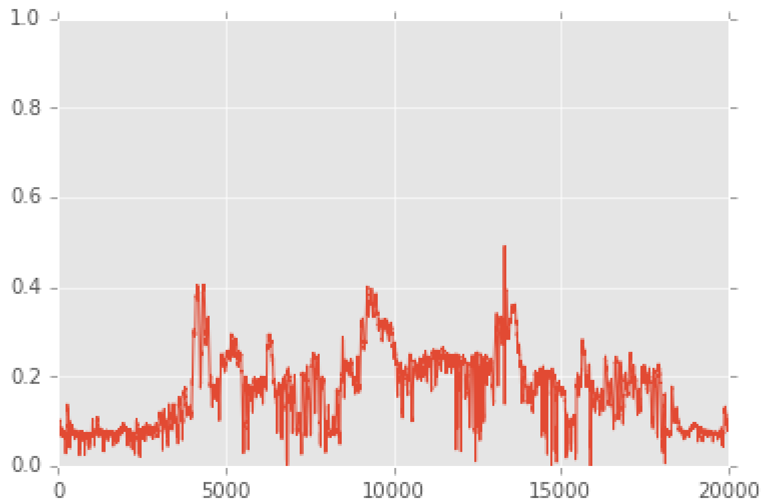
```
Test Score: 0.16 RMSE
```

It seems that the network is having a hard time predicting higher flood level values

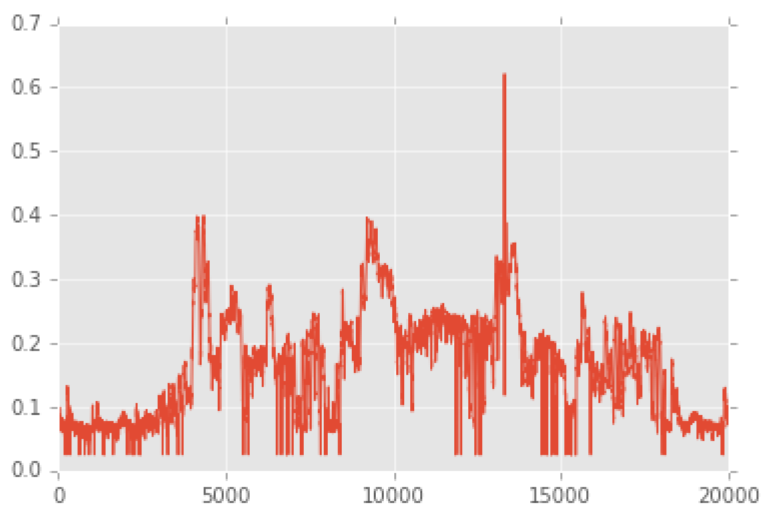
Checking the first 20000 data shows that the network is relatively comfortable on predicting ahead of time flood level values when the flood level aren't extreme

In [119]:
```
plt.gca().set_ylim(bottom=0)
plt.gca().set_ylim(top=1)

print('----------------- Predicted ---------------')
plt.plot(unseen_results[0:20000])
plt.show()

print('----------------- Ground Truth ---------------')
plt.plot(labels[0:20000])
plt.show()
```

```
----------------- Predicted ---------------
```




```
----------------- Ground Truth ---------------
```



In [ ]: