Deep Learning

Anomalies in Streaming Data

U Kang
Seoul National University

In This Lecture

- Streaming (or time-series) data
- Anomaly detection
- Recurrent neural networks
- Stacked LSTM structure

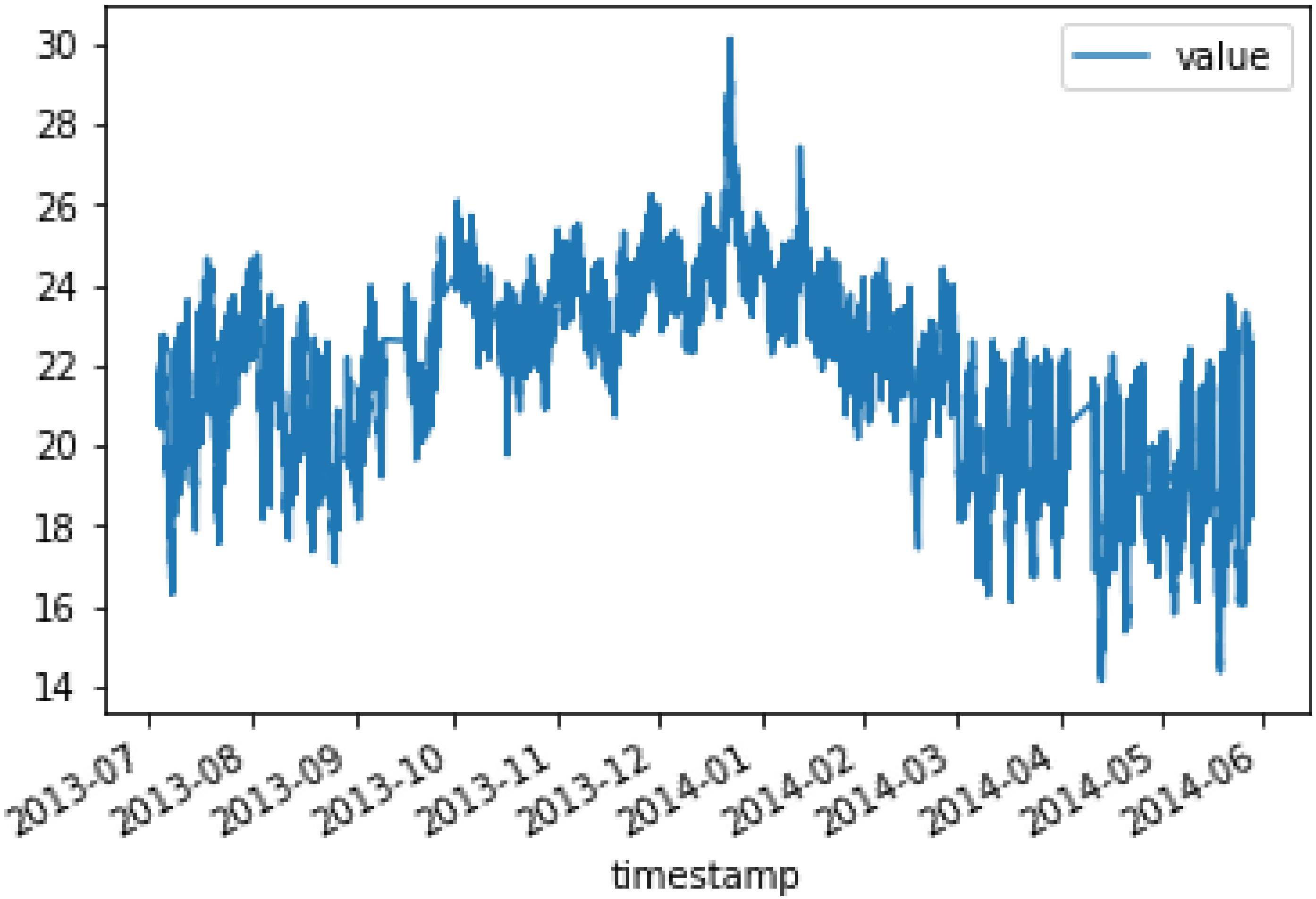
Outline

- ☐ **Problem Definition**
- ☐ Requirements
- ☐ Preprocessing Codes
- ☐ Answers

Dataset (1)

- Numenta Anomaly Benchmark (NAB)
  - Benchmark for anomaly detection in streaming
  - Composed of over 50 time-series data files
  - Data are timestamped and single-valued metrics
- We use one of the included datasets:
  - The ambient temperature in an office setting

Dataset (2)



value
30
28
26
24
22
20
18
16
14
2013-07
2013-08
2013-09
2013-10
2013-11
2013-12
2014-01
2014-02
2014-03
2014-04
2014-05
2014-06
timestamp

Problem Definition

- To find anomalies in time-series data
- What are anomalies?
  - Data points that follow abnormal patterns
- Unsupervised problem
  - No explicit labels (or answers)

How to Solve

- Train a model for time-series prediction
- Predict the values (via regression)
- Compute the prediction errors for all points
- Pick k points with the largest differences

- We classify these points as anomalies!
  - Because they are unpredictable from the model

Selection of an Algorithm

- We have a time-series dataset
- RNN (recurrent neural network) will be good
- Especially, we use the LSTM structure

Outline

- ☑ Problem Definition
- ☐ **Requirements**
- ☐ Preprocessing Codes
- ☐ Answers

Feature Engineering

- The dataset is single-valued
  - Contains (time, value) for each timestamp
- We need more features for high performance
- Create **at least 4 features**
  - For instance, HOUR of each timestamp

Feature Distribution

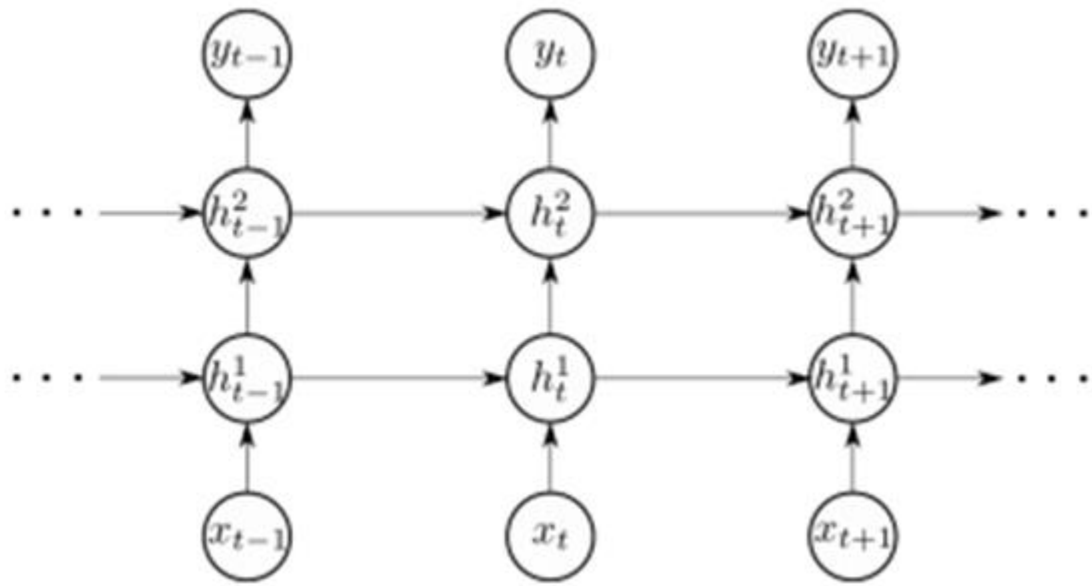
- The features need to have similar distributions
- Thus, standardize them by
  - Setting the mean to 0
  - Scaling to unit variance
- It is helpful for most ML algorithms

Unrolling Data

- LSTM takes a times-series of arbitrary length
- But, too long sequences are not very helpful
- We limit the number of LSTM cells to 50
  - That is, we use 50 historical values for prediction

Model Structure (1)

- Implement a stacked LSTM with 2 layers



Model Structure (2)

- Set output dimensions of the LSTM as
  - 50 in the first layer
  - 100 in the second layer
- Add dropout after each layer
- Add a dense layer to produce the prediction
- Use the MSE loss with Adam optimizer

Outline

- ☑ Problem Definition
- ☑ Requirements
- ☐ **Preprocessing Codes**
- ☐ Answers

Importing Packages

- Import necessary packages:

```
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn import preprocessing
from matplotlib import pyplot as plt
```

Reading the Dataset (1)

- Read the dataset using pandas:

```
df = pd.read_csv('data/ambient_temperature_system_failure.csv')
df.head()
```

	timestamp	value
0	2013-07-04 00:00:00	69.880835
1	2013-07-04 01:00:00	71.220227
2	2013-07-04 02:00:00	70.877805
3	2013-07-04 03:00:00	68.959400
4	2013-07-04 04:00:00	69.283551

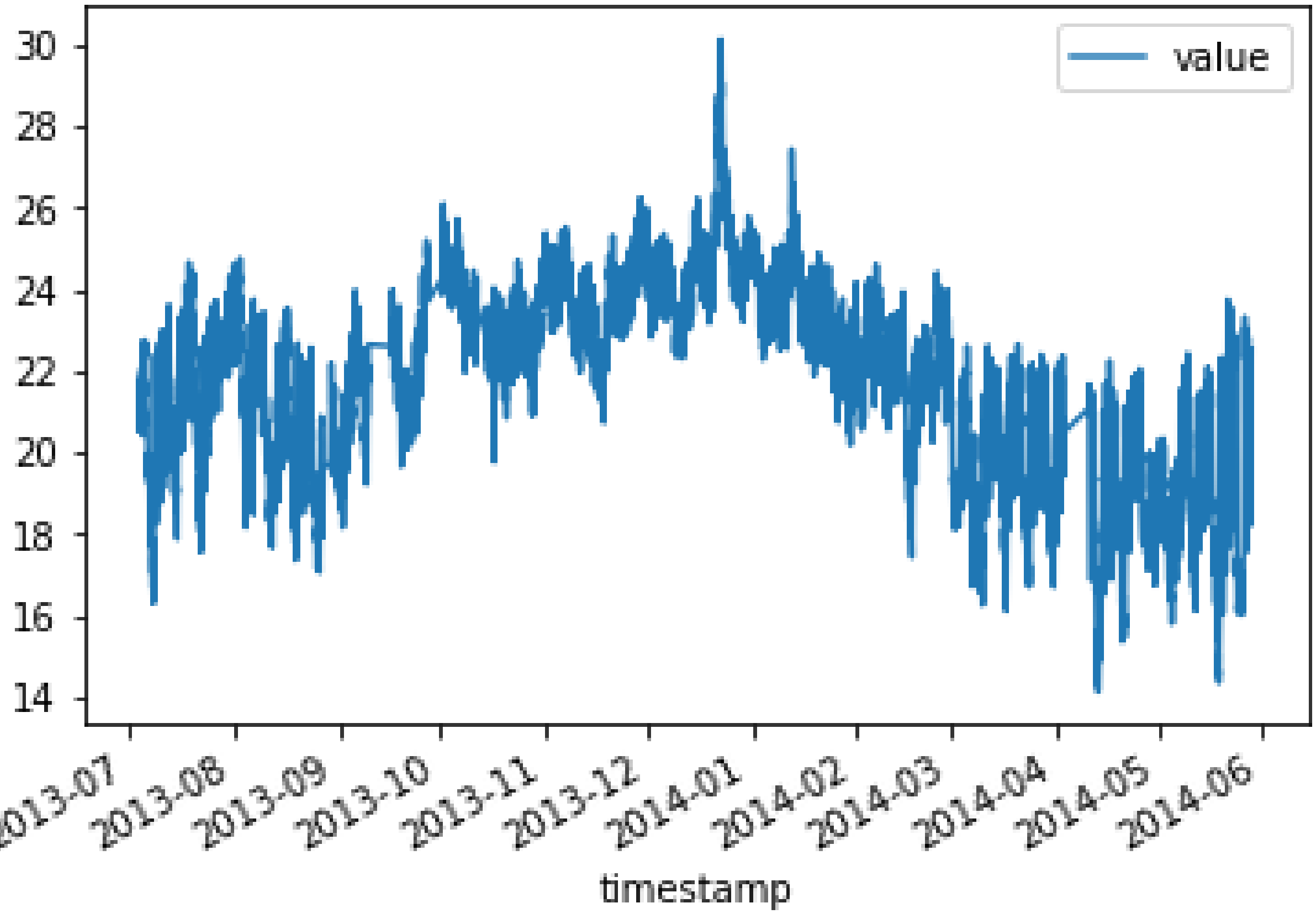
Reading the Dataset (2)

- Modify the dataset and plot the values:

```
df['timestamp'] = pd.to_datetime(df['timestamp'])
df['value'] = (df['value'] - 32) * 5 / 9
df.plot(x='timestamp', y='value')
```

- Change the type of timestamp column (line 1)
- Change Fahrenheit to Celcius (line 2)
- Plot the figure (line 3)

Dataset (2)



30
28
26
24
22
20
18
16
14
value
2013-07
2013-08
2013-09
2013-10
2013-11
2013-12
2014-01
2014-02
2014-03
2014-04
2014-05
2014-06
timestamp

Creating Features

- Create four additional features:

```
df['hours'] = df['timestamp'].dt.hour
df['daylight'] = ((df['hours'] >= 7) &
                  (df['hours'] <= 22)).astype(int)
df['dayofweek'] = df['timestamp'].dt.dayofweek
df['weekday'] = (df['dayofweek'] < 5).astype(int)
```

- hours: the hour of each timestamp
- daylight: whether it is daytime or not
- dayofweek: day of the week
- weekday: whether it is a weekday or not

Standardizing the Features

- Standardize the added features:

```
data_n = df[['value', 'hours', 'daylight',
             'dayofweek', 'weekday']]
type(data_n)
std_scaler = preprocessing.StandardScaler()
scaled = std_scaler.fit_transform(data_n)
data_n = pd.DataFrame(scaled)
```

- data_n is a DataFrame with 5 features

Unrolling the Features

- We unroll the features with length 50:

```
#unroll: create sequence of 50 previous data points for each data points
def unroll(data,length=50):
    result = []
    for i in range(len(data) - length + 1):
        result.append(data[i : i + length])
    return np.asarray(result)
```

```
X = data_n[:-1].values
X = unroll(X, length = 50)
print(f"X shape: {X.shape}")
```

```
X shape: (7217, 50, 5)
```

Creating Labels

- We create labels that we want to predict:

```
y = data_n[1:][0].values
y = y[-X.shape[0]:]

print(f"Y shape: {y.shape}")
```

```
Y shape: (7217,)
```

- Note that x and y have the same length

Dividing Instances (1)

- Create training and test sets:

```
test_size = 1000

X_train = X[:-test_size]
X_test = X[-test_size:]

y_train = y[:-test_size]
y_test = y[-test_size:]
```

Dividing Instances (2)

- Check their shapes:

```
# see the shape
print(f"x_train: {X_train.shape} "
      f"y_train: {y_train.shape}")

print(f"x_test : {X_test.shape} "
      f"y_test : {y_test.shape}")
```

```
x_train: (6217, 50, 5) y_train: (6217,)
x_test : (1000, 50, 5) y_test : (1000,)
```

- Note that they are already unrolled

Helpful Modules

- You may need these modules:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Outline

- ☑ Problem Definition
- ☑ Requirements
- ☑ Preprocessing Codes
- ☐ **Answers**

Define the model

- Define the model with helpful modules
  - They have different numbers of units
  - We apply dropout for each layer

```
def create_lstm(pkeep):
    # Build the model
    model = Sequential()

    model.add(LSTM(50, return_sequences = True))
    model.add(Dropout(1-pkeep))

    model.add(LSTM(100))
    model.add(Dropout(1-pkeep))

    model.add(Dense(units=1))

    return model
```

Set Hyperparameters

- Set parameters for learning:

```
pkeep = 0.8
batch_size = 512
epochs = 10
```

- Create the model with `create_lstm()`
- Define cost and optimizer variables
  - loss function: mean squared error
  - optimizer : Adam

```
model = create_lstm(pkeep)
model.compile(loss = 'mse', optimizer = 'adam')
```

Train the Model

- Learn the model's parameters
- Early stopping
  - Use 10% of train data set as validation data

```
# Train the model
model.fit(X_train, y_train, batch_size=batch_size,
          epochs=epochs, validation_split=0.1,
          callbacks=[EarlyStopping(monitor="val_loss", patience=10)])
```

```
Epoch 1/20
11/11 [==============================] - 8s 332ms/step - loss: 0.7556
- val_loss: 0.3033
Epoch 2/20
11/11 [==============================] - 3s 251ms/step - loss: 0.2339
- val_loss: 0.1999
Epoch 3/20
11/11 [==============================] - 3s 257ms/step - loss: 0.1803
- val_loss: 0.1964
```

Test Differences

- Compute the differences for the test instances:

```
preds = model.predict(X_test)
diffs = list(abs(y_test - preds.flatten()))
```

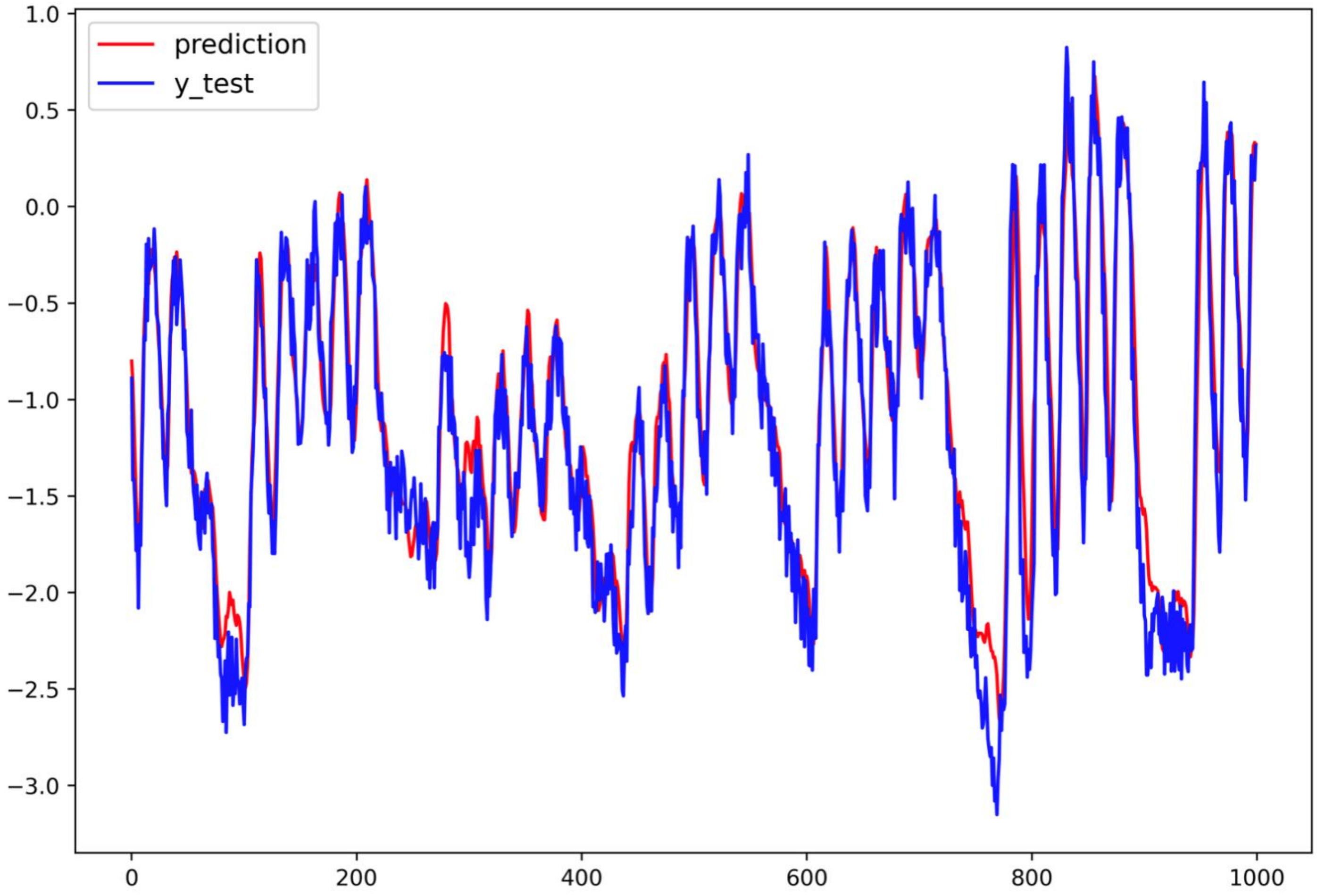
- Here a difference is a simple $|x - x'|$ form

True Data vs. Predictions (1)

- Plot the true data and predictions:

```
fig, axs = plt.subplots()
axs.plot(preds, color='red', label='prediction')
axs.plot(y_test, color='blue', label='y_test')
plt.legend(loc='upper left')
plt.show()
```

True Data vs. Predictions (2)



Finding Anomalies

- Set the number of outliers to 10
- Find 10 points with maximum differences:

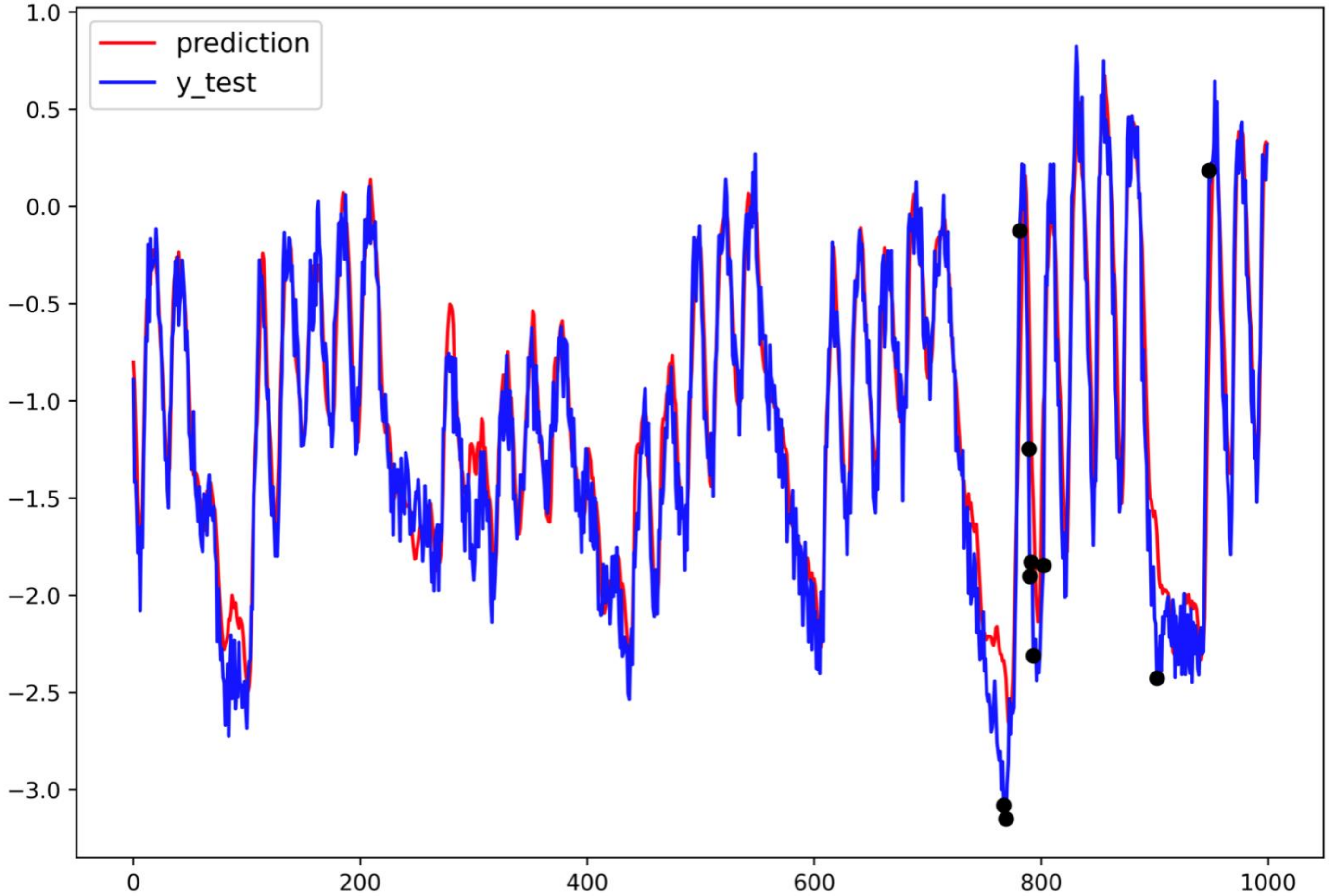
```
n_outliers = 10
argsorted = np.array(diffs).argsort()
anomalies = argsorted[-n_outliers:][::-1]
```

Plotting the Anomalies (1)

- Plot the anomalies in the previous figure:

```
fig, axs = plt.subplots()
axs.plot(preds, color='red', label='prediction')
axs.plot(y_test, color='blue', label='y_test')
axs.scatter(anomalies, [y_test[i] for i in anomalies],
            color='black', zorder=10)
plt.legend(loc='upper left')
plt.show()
```

Plotting the Anomalies (2)



What You Need to Know

- Streaming (or time-series) data
- Anomaly detection
- Recurrent neural networks
- Stacked LSTM structure

Questions?