# **基于LSTM及其变种网络的时序数据建模实战教程**

**单变量LSTM模型  
多元LSTM模型  
多步骤LSTM模型  
多元多步LSTM模型**

1. **单向LSTM**

**# univariate lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense**  
 **# split a univariate sequence into samples  
def split\_sequence(sequence, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the sequence  
        if end\_ix > len(sequence)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps = 3  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps)  
# reshape from [samples, timesteps] into [samples, timesteps, features]  
n\_features = 1  
X = X.reshape((X.shape[0], X.shape[1], n\_features))  
# define model  
model = Sequential()  
model.add(LSTM(50, activation='relu', input\_shape=(n\_steps, n\_features)))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=200, verbose=0)  
# demonstrate prediction  
x\_input = array([70, 80, 90])  
x\_input = x\_input.reshape((1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**2.堆叠LSTM**

**# univariate stacked lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
  
# split a univariate sequence  
def split\_sequence(sequence, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the sequence  
        if end\_ix > len(sequence)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps = 3  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps)  
# reshape from [samples, timesteps] into [samples, timesteps, features]  
n\_features = 1  
X = X.reshape((X.shape[0], X.shape[1], n\_features))  
# define model  
model = Sequential()  
model.add(LSTM(50, activation='relu', return\_sequences=True, input\_shape=(n\_steps, n\_features)))  
model.add(LSTM(50, activation='relu'))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=200, verbose=0)  
# demonstrate prediction  
x\_input = array([70, 80, 90])  
x\_input = x\_input.reshape((1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**3.Vanilla LSTM**

**# univariate lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
  
# split a univariate sequence into samples  
def split\_sequence(sequence, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the sequence  
        if end\_ix > len(sequence)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps = 3  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps)  
# reshape from [samples, timesteps] into [samples, timesteps, features]  
n\_features = 1  
X = X.reshape((X.shape[0], X.shape[1], n\_features))  
# define model  
model = Sequential()  
model.add(LSTM(50, activation='relu', input\_shape=(n\_steps, n\_features)))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=200, verbose=0)  
# demonstrate prediction  
x\_input = array([70, 80, 90])  
x\_input = x\_input.reshape((1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**4.双向LSTM**

**# univariate bidirectional lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
from keras.layers import Bidirectional  
  
# split a univariate sequence  
def split\_sequence(sequence, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the sequence  
        if end\_ix > len(sequence)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps = 3  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps)  
# reshape from [samples, timesteps] into [samples, timesteps, features]  
n\_features = 1  
X = X.reshape((X.shape[0], X.shape[1], n\_features))  
# define model  
model = Sequential()  
model.add(Bidirectional(LSTM(50, activation='relu'), input\_shape=(n\_steps, n\_features)))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=200, verbose=0)  
# demonstrate prediction  
x\_input = array([70, 80, 90])  
x\_input = x\_input.reshape((1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**5.卷积LSTM**

**# univariate cnn lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
from keras.layers import Flatten  
from keras.layers import TimeDistributed  
from keras.layers.convolutional import Conv1D  
from keras.layers.convolutional import MaxPooling1D  
  
# split a univariate sequence into samples  
def split\_sequence(sequence, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the sequence  
        if end\_ix > len(sequence)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps = 4  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps)  
# reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]  
n\_features = 1  
n\_seq = 2  
n\_steps = 2  
X = X.reshape((X.shape[0], n\_seq, n\_steps, n\_features))  
# define model  
model = Sequential()  
model.add(TimeDistributed(Conv1D(filters=64, kernel\_size=1, activation='relu'), input\_shape=(None, n\_steps, n\_features)))  
model.add(TimeDistributed(MaxPooling1D(pool\_size=2)))  
model.add(TimeDistributed(Flatten()))  
model.add(LSTM(50, activation='relu'))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=500, verbose=0)  
# demonstrate prediction  
x\_input = array([60, 70, 80, 90])  
x\_input = x\_input.reshape((1, n\_seq, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

1. **ConvLSTM**

**# univariate convlstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
from keras.layers import Flatten  
from keras.layers import ConvLSTM2D  
  
# split a univariate sequence into samples  
def split\_sequence(sequence, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the sequence  
        if end\_ix > len(sequence)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps = 4  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps)  
# reshape from [samples, timesteps] into [samples, timesteps, rows, columns, features]  
n\_features = 1  
n\_seq = 2  
n\_steps = 2  
X = X.reshape((X.shape[0], n\_seq, 1, n\_steps, n\_features))  
# define model  
model = Sequential()  
model.add(ConvLSTM2D(filters=64, kernel\_size=(1,2), activation='relu', input\_shape=(n\_seq, 1, n\_steps, n\_features)))  
model.add(Flatten())  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=500, verbose=0)  
# demonstrate prediction  
x\_input = array([60, 70, 80, 90])  
x\_input = x\_input.reshape((1, n\_seq, 1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**7.多变量LSTM**

**（1）**

**# multivariate lstm example  
from numpy import array  
from numpy import hstack  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
  
# split a multivariate sequence into samples  
def split\_sequences(sequences, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequences)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the dataset  
        if end\_ix > len(sequences):  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequences[i:end\_ix, :-1], sequences[end\_ix-1, -1]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
in\_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])  
in\_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])  
out\_seq = array([in\_seq1[i]+in\_seq2[i] for i in range(len(in\_seq1))])  
# convert to [rows, columns] structure  
in\_seq1 = in\_seq1.reshape((len(in\_seq1), 1))  
in\_seq2 = in\_seq2.reshape((len(in\_seq2), 1))  
out\_seq = out\_seq.reshape((len(out\_seq), 1))  
# horizontally stack columns  
dataset = hstack((in\_seq1, in\_seq2, out\_seq))  
# choose a number of time steps  
n\_steps = 3  
# convert into input/output  
X, y = split\_sequences(dataset, n\_steps)  
# the dataset knows the number of features, e.g. 2  
n\_features = X.shape[2]  
# define model  
model = Sequential()  
model.add(LSTM(50, activation='relu', input\_shape=(n\_steps, n\_features)))  
model.add(Dense(1))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=200, verbose=0)  
# demonstrate prediction  
x\_input = array([[80, 85], [90, 95], [100, 105]])  
x\_input = x\_input.reshape((1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**（2）**

**# multivariate output stacked lstm example  
from numpy import array  
from numpy import hstack  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
  
# split a multivariate sequence into samples  
def split\_sequences(sequences, n\_steps):  
    X, y = list(), list()  
    for i in range(len(sequences)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps  
        # check if we are beyond the dataset  
        if end\_ix > len(sequences)-1:  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequences[i:end\_ix, :], sequences[end\_ix, :]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
in\_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])  
in\_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])  
out\_seq = array([in\_seq1[i]+in\_seq2[i] for i in range(len(in\_seq1))])  
# convert to [rows, columns] structure  
in\_seq1 = in\_seq1.reshape((len(in\_seq1), 1))  
in\_seq2 = in\_seq2.reshape((len(in\_seq2), 1))  
out\_seq = out\_seq.reshape((len(out\_seq), 1))  
# horizontally stack columns  
dataset = hstack((in\_seq1, in\_seq2, out\_seq))  
# choose a number of time steps  
n\_steps = 3  
# convert into input/output  
X, y = split\_sequences(dataset, n\_steps)  
# the dataset knows the number of features, e.g. 2  
n\_features = X.shape[2]  
# define model  
model = Sequential()  
model.add(LSTM(100, activation='relu', return\_sequences=True, input\_shape=(n\_steps, n\_features)))  
model.add(LSTM(100, activation='relu'))  
model.add(Dense(n\_features))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=400, verbose=0)  
# demonstrate prediction  
x\_input = array([[70,75,145], [80,85,165], [90,95,185]])  
x\_input = x\_input.reshape((1, n\_steps, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**多步长**

**# univariate multi-step vector-output stacked lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
  
# split a univariate sequence into samples  
def split\_sequence(sequence, n\_steps\_in, n\_steps\_out):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps\_in  
        out\_end\_ix = end\_ix + n\_steps\_out  
        # check if we are beyond the sequence  
        if out\_end\_ix > len(sequence):  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix:out\_end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps\_in, n\_steps\_out = 3, 2  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps\_in, n\_steps\_out)  
# reshape from [samples, timesteps] into [samples, timesteps, features]  
n\_features = 1  
X = X.reshape((X.shape[0], X.shape[1], n\_features))  
# define model  
model = Sequential()  
model.add(LSTM(100, activation='relu', return\_sequences=True, input\_shape=(n\_steps\_in, n\_features)))  
model.add(LSTM(100, activation='relu'))  
model.add(Dense(n\_steps\_out))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=50, verbose=0)  
# demonstrate prediction  
x\_input = array([70, 80, 90])  
x\_input = x\_input.reshape((1, n\_steps\_in, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**encode-decode模型**

**# univariate multi-step encoder-decoder lstm example  
from numpy import array  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
from keras.layers import RepeatVector  
from keras.layers import TimeDistributed  
  
# split a univariate sequence into samples  
def split\_sequence(sequence, n\_steps\_in, n\_steps\_out):  
    X, y = list(), list()  
    for i in range(len(sequence)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps\_in  
        out\_end\_ix = end\_ix + n\_steps\_out  
        # check if we are beyond the sequence  
        if out\_end\_ix > len(sequence):  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequence[i:end\_ix], sequence[end\_ix:out\_end\_ix]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
raw\_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]  
# choose a number of time steps  
n\_steps\_in, n\_steps\_out = 3, 2  
# split into samples  
X, y = split\_sequence(raw\_seq, n\_steps\_in, n\_steps\_out)  
# reshape from [samples, timesteps] into [samples, timesteps, features]  
n\_features = 1  
X = X.reshape((X.shape[0], X.shape[1], n\_features))  
y = y.reshape((y.shape[0], y.shape[1], n\_features))  
# define model  
model = Sequential()  
model.add(LSTM(100, activation='relu', input\_shape=(n\_steps\_in, n\_features)))  
model.add(RepeatVector(n\_steps\_out))  
model.add(LSTM(100, activation='relu', return\_sequences=True))  
model.add(TimeDistributed(Dense(1)))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=100, verbose=0)  
# demonstrate prediction  
x\_input = array([70, 80, 90])  
x\_input = x\_input.reshape((1, n\_steps\_in, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**

**多路并行输入和多步输出**

**# multivariate multi-step encoder-decoder lstm example  
from numpy import array  
from numpy import hstack  
from keras.models import Sequential  
from keras.layers import LSTM  
from keras.layers import Dense  
from keras.layers import RepeatVector  
from keras.layers import TimeDistributed  
  
# split a multivariate sequence into samples  
def split\_sequences(sequences, n\_steps\_in, n\_steps\_out):  
    X, y = list(), list()  
    for i in range(len(sequences)):  
        # find the end of this pattern  
        end\_ix = i + n\_steps\_in  
        out\_end\_ix = end\_ix + n\_steps\_out  
        # check if we are beyond the dataset  
        if out\_end\_ix > len(sequences):  
            break  
        # gather input and output parts of the pattern  
        seq\_x, seq\_y = sequences[i:end\_ix, :], sequences[end\_ix:out\_end\_ix, :]  
        X.append(seq\_x)  
        y.append(seq\_y)  
    return array(X), array(y)  
  
# define input sequence  
in\_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])  
in\_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])  
out\_seq = array([in\_seq1[i]+in\_seq2[i] for i in range(len(in\_seq1))])  
# convert to [rows, columns] structure  
in\_seq1 = in\_seq1.reshape((len(in\_seq1), 1))  
in\_seq2 = in\_seq2.reshape((len(in\_seq2), 1))  
out\_seq = out\_seq.reshape((len(out\_seq), 1))  
# horizontally stack columns  
dataset = hstack((in\_seq1, in\_seq2, out\_seq))  
# choose a number of time steps  
n\_steps\_in, n\_steps\_out = 3, 2  
# covert into input/output  
X, y = split\_sequences(dataset, n\_steps\_in, n\_steps\_out)  
# the dataset knows the number of features, e.g. 2  
n\_features = X.shape[2]  
# define model  
model = Sequential()  
model.add(LSTM(200, activation='relu', input\_shape=(n\_steps\_in, n\_features)))  
model.add(RepeatVector(n\_steps\_out))  
model.add(LSTM(200, activation='relu', return\_sequences=True))  
model.add(TimeDistributed(Dense(n\_features)))  
model.compile(optimizer='adam', loss='mse')  
# fit model  
model.fit(X, y, epochs=300, verbose=0)  
# demonstrate prediction  
x\_input = array([[60, 65, 125], [70, 75, 145], [80, 85, 165]])  
x\_input = x\_input.reshape((1, n\_steps\_in, n\_features))  
yhat = model.predict(x\_input, verbose=0)  
print(yhat)**