# 使用LSTM做2d序列预测

这里专门再做一次序列预测的总结，是为了突出LSTM输入数据的shape设置方法。LSTM在网络定义的时候，input\_shape应设置为（time\_step, features），其中time\_step也就是look\_back，即输入序列的长度，也就是考察多长的时间点，features为每个时间点上观察到的特征数量，这里是以2d序列为例，观察到的是点的x和y坐标，因此features应设置为2。在喂给网络数据时，数据的维度必须是3维的，即(samples, time\_step, features)，分别表示所有样本的数量、时间步、特征数。

samples是所有样本的数量，batch\_size是一次性计算多少组样本，每一个样本又是由time\_step个时间点的状态组成，每一个状态有features个特征。

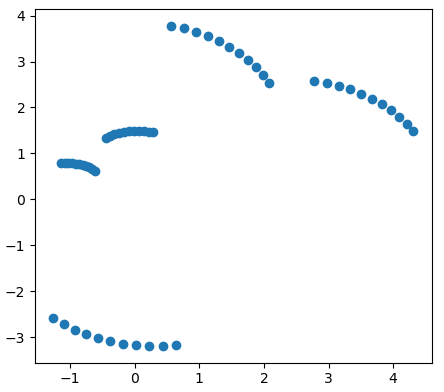
## 数据准备

这里我们用代码生成1000圆弧形样本，每个样本包括11个点，我们用前10个点来预测第11个点，每个点给出x和y坐标，因此，我们的features为2，time\_step(look\_back)为10，samples为1000。

|  |
| --- |
| **def** create\_dataset(samples\_num, look\_back):  seqs = []  **for** i **in** range (samples\_num):  a, r = np.random.rand(2)  a = a \* 6.28 - 3.14  r = r \* 3.0  x, y = np.random.rand(2) \* 5 - 2.5  seq = []  **for** j **in** range(look\_back + 1):  a = a + 0.087  px = x + r \* cos(a)  py = y + r \* sin(a)  seq.append([px, py])  seqs.append(seq)  seqs = np.array(seqs)  **return** seqs  *#create dataset* dataset = create\_dataset(samples\_num, look\_back) |

我们使用以下代码来观察前5个样本：

|  |
| --- |
| plt.scatter(dataset[0:5, :, 0], dataset[0:5, :, 1]) plt.show() |



我们将数据归一化，并分成训练样本和测试样本：

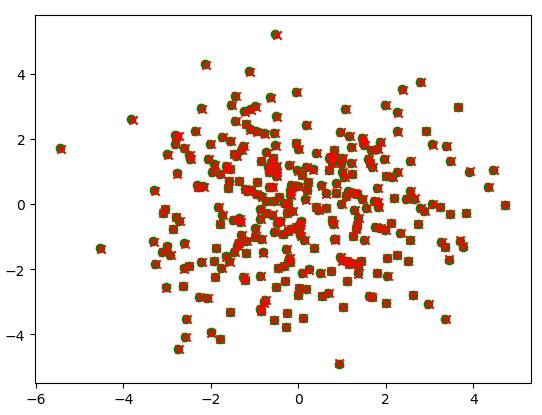
|  |
| --- |
| *#transform to (0, 1)* scaler = MinMaxScaler(feature\_range=(0, 1)) data\_shape = dataset.shape dataset = dataset.reshape(-1, 1) dataset = scaler.fit\_transform(dataset) dataset = dataset.reshape(data\_shape) dataX = dataset[:, :-1, :] dataY = dataset[:, -1, :] *#split dataset to train and test* trainX = dataX[:int(samples\_num \* 0.7), :, :] trainY = dataY[:int(samples\_num \* 0.7), :] testX = dataX[int(samples\_num\*0.7):, :, :] testY = dataY[int(samples\_num\*0.7):, :] |

## 搭建并训练网络

|  |
| --- |
| **def** trainNet(trainX, trainY, epochs, batch\_size):  model = Sequential()  model.add(LSTM(10, input\_shape=(10, 2)))  model.add(Dense(2))  model.compile(loss=**'mean\_squared\_error'**, optimizer=**'adam'**)  model.fit(trainX, trainY, epochs = epochs, batch\_size=batch\_size)  model.save(**'rnn\_2d.h5'**)  **return** model |

## 预测

|  |
| --- |
| *#predict* predictY = model.predict(testX) predictY = scaler.inverse\_transform(predictY) testY = scaler.inverse\_transform(testY)  *#draw* plt.scatter(testY[:, 0], testY[:, 1], color=**'green'**) plt.scatter(predictY[:,0], predictY[:,1], color=**'red'**, marker=**'x'**) plt.show() |



绿色圆点为标签，红色叉号为预测结果。

## 代码

|  |
| --- |
| **import** numpy **as** np **from** math **import** \* **import** matplotlib.pyplot **as** plt **from** keras **import** Sequential **from** keras.layers **import** LSTM **from** keras.layers **import** Dense **from** keras.models **import** load\_model **from** sklearn.preprocessing **import** MinMaxScaler  TRAIN\_MODE = 0  **def** create\_dataset(samples\_num, look\_back):  seqs = []  **for** i **in** range (samples\_num):  a, r = np.random.rand(2)  a = a \* 6.28 - 3.14  r = r \* 3.0  x, y = np.random.rand(2) \* 5 - 2.5  seq = []  **for** j **in** range(look\_back + 1):  a = a + 0.087  px = x + r \* cos(a)  py = y + r \* sin(a)  seq.append([px, py])  seqs.append(seq)  seqs = np.array(seqs)  **return** seqs  **def** trainNet(trainX, trainY, epochs, batch\_size):  model = Sequential()  model.add(LSTM(10, input\_shape=(10, 2)))  model.add(Dense(2))  model.compile(loss=**'mean\_squared\_error'**, optimizer=**'adam'**)  model.fit(trainX, trainY, epochs = epochs, batch\_size=batch\_size)  model.save(**'rnn\_2d.h5'**)  **return** model  **def** loadNet():  model = load\_model(**'rnn\_2d.h5'**)  **return** model  samples\_num = 1000 look\_back = 10 *#create dataset* dataset = create\_dataset(samples\_num, look\_back)  plt.scatter(dataset[0:5, :, 0], dataset[0:5, :, 1]) plt.show()  *#transform to (0, 1)* scaler = MinMaxScaler(feature\_range=(0, 1)) data\_shape = dataset.shape dataset = dataset.reshape(-1, 1) dataset = scaler.fit\_transform(dataset) dataset = dataset.reshape(data\_shape) dataX = dataset[:, :-1, :] dataY = dataset[:, -1, :] *#split dataset to train and test* trainX = dataX[:int(samples\_num \* 0.7), :, :] trainY = dataY[:int(samples\_num \* 0.7), :] testX = dataX[int(samples\_num\*0.7):, :, :] testY = dataY[int(samples\_num\*0.7):, :] *#train or load model* **if** TRAIN\_MODE:  model = trainNet(trainX, trainY, 500, 10) **else**:  model = loadNet() *#predict* predictY = model.predict(testX) predictY = scaler.inverse\_transform(predictY) testY = scaler.inverse\_transform(testY)  *#draw* plt.scatter(testY[:, 0], testY[:, 1], color=**'green'**) plt.scatter(predictY[:,0], predictY[:,1], color=**'red'**, marker=**'x'**) plt.show() |