认知科学与类脑计算

实验报告四

日期：2019/5/24

班级：16 人工智能

姓名：贾乘兴

学号：201600301304

一.需求分析：

LSTM（Long Short-Term Memory）是长短期记忆网络，是一种时间递归神经网络，适合于处理和预测时间序列中间隔和延迟相对较长的重要事件。LSTM 区别于 RNN 的地方，主要就在于它在算法中加入了一个判断信息有用与否的“处理器”，这个处理器作用的结构被称为 cell。一个 cell 当中被放置了三扇门，分别叫做输入门、遗忘门和输出门。一个信息进入 LSTM 的网络当中，可以根据规则来判断是否有用。只有符合算法认证的信息才会留下，不符的信息则通过遗忘门被遗忘。说起来无非就是一进二出的工作原理，却可以在反复运算下解决神经网络中长期存在的大问题。目前已经证明，LSTM 是解决长序依赖问题的有效技术，并且这种技术的普适性非常高，导致带来的可能性变化非常多。各研究者根据 LSTM 纷纷提出了自己的变量版本，这就让 LSTM 可以处理千变万化的垂直问题。本次实验的目的是加深对 LSTM 模型的理解，能够使用 LSTM 模型解决简单问题

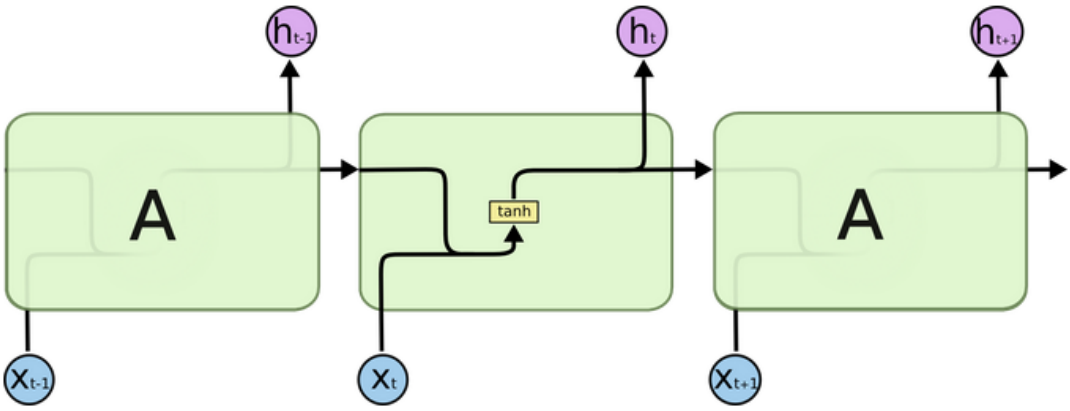
二.概要设计：

随机产生 0-127 之间的两个八位的二进制整数，作为一组输入数据，将这两个数的和作为一个标签，这三个数据组成一组训练数据，训练数据的组数应尽可能多。然后创建 LSTM 网络（手动实现和基于 keras 实现）。实现两个八位的二进制整数的加法运算，网络能够输出正确的加法运算结果。

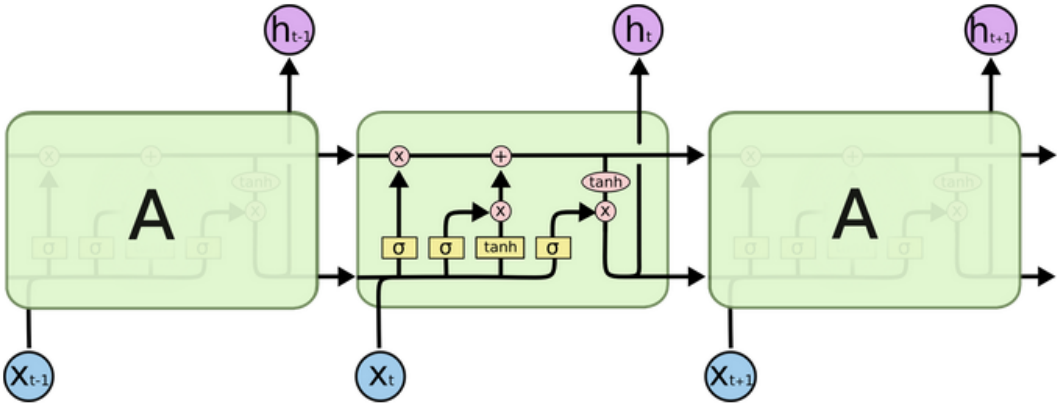
三.详细设计：

1. LSTM 模型

LSTM 是一类可以处理长期依赖问题的特殊的 RNN，LSTM 主要用来处理长期依赖问题，与传统 RNN 相比，长时间的信息记忆能力是与生俱来的。所有的 RNN 链式结构中都有不断重复的模块，用来随时间传递信息。传统的 RNN 使用十分简单的结构，如下图所示。



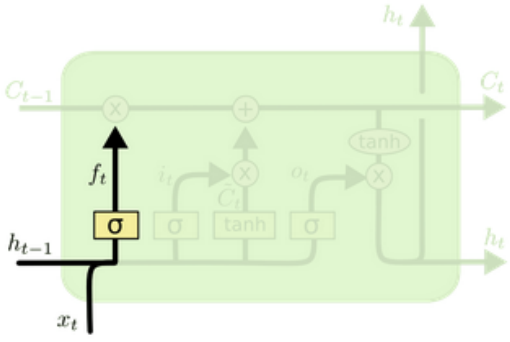
LSTM 链式结构中重复模块的结构更加复杂，有四个互相交互的层 (如下图所示)。



与传统 RNN 相比，除了拥有隐藏状态外，LSTM 还增加了一个细胞状态，记录随时间传递的信息。在传递过程中，通过当前输入、上一时刻隐藏层状态、上一时刻细胞状态以及门结构来增加或删除细胞状态中的信息。门结构用来控制增加或删除信息的程度，一般由 sigmoid 函数和向量点乘来实现。

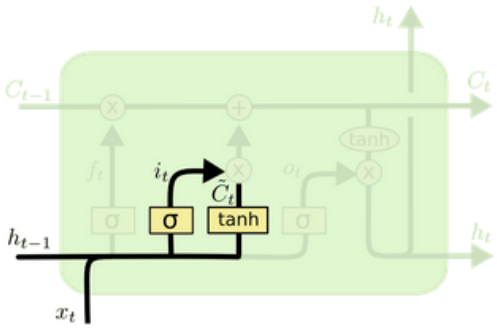
LSTM 共包含 3 个门结构，来控制细胞状态和隐藏状态，下边分别进行介绍。

遗忘门。遗忘门决定上一时刻细胞状态中的多少信息可以传递到当前时刻中。



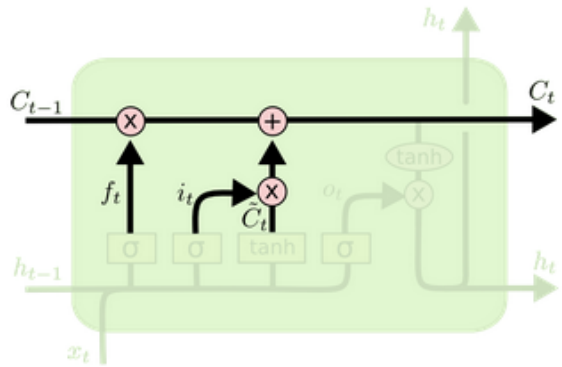
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

输入门。输入门用来控制当前输入新生成的信息中有多少信息可以加入到细胞状态中。C_t层用来产生当前时刻新的信息，i_t层用来控制有多少新信息可以传递给细胞状态。



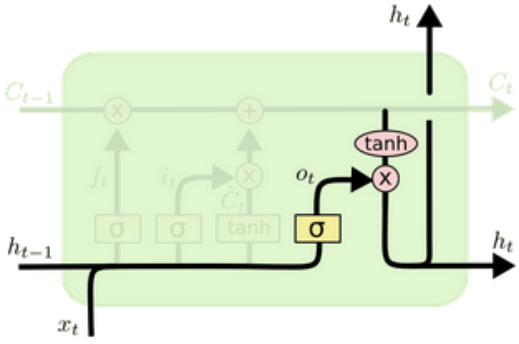
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

更新细胞状态。基于遗忘门和输入门的输出，来更新细胞状态。更新后的细胞状态有两部分构成:一，来自上一时刻旧的细胞状态信息；二，当前输入新生成的信息。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

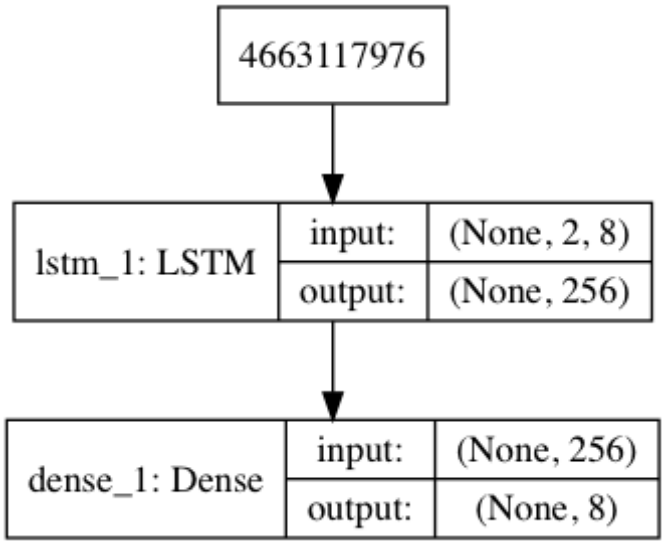
输出门，基于更新的细胞状态，输出隐藏状态。



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

2. 神经网络设计

首先是将两个二进制数字输入 lstm 中，在最后输出的 h 状态中，我们在加入一层全联接层，将隐藏层状态映射到八维的向量上，整体网络设计如下



在输出层使用了 sigmoid 函数，映射至 01 之间

3. 损失函数、正确率、预测设计

损失函数使用的是均方误差

$$l=\frac{1}{N\cdot n}\sum_{i=1}^{N}\left(y_i-t_i\right)^T\left(y_i-t_i\right)$$

其中 n 是向量维度，N 是样本数

在训练时测试集的正确率我们不设阈值，完全比较向量的预测

在预测时，我们设置阈值为 0.5，将向量二值化，然后计算数值相差

3. 损失函数权重

在手写的 lstm 版本中，是将离散样本空间变为独热的数据，所以每个部分对结果的贡献是一样的，但在二维数字中，每个位置的数值贡献是不一样的，所以按照其对应的数值设置权重，即

$$l=\frac{1}{N\cdot n}\sum_{i=1}^{N}\left[a\odot\left(y_i-t_i\right)^T\right]\left(y_i-t_i\right)$$
$$a=[128,64,32,16,8,4,2,1]$$

四.调试分析：

在不断增加样本的过程中，对计算结果的准确度越来越高，在只有 100 个样本的计算结果如下

```
num0 : 95 + 120 = 215, lstm(95,120) = 249
num1 : 18 + 78 = 96, lstm(18,78) = 114
num2 : 28 + 55 = 83, lstm(28,55) = 83
num3 : 36 + 53 = 89, lstm(36,53) = 83
num4 : 60 + 90 = 150, lstm(60,90) = 130
num5 : 8 + 87 = 95, lstm(8,87) = 83
num6 : 80 + 59 = 139, lstm(80,59) = 131
num7 : 36 + 72 = 108, lstm(36,72) = 82
num8 : 66 + 34 = 100, lstm(66,34) = 126
num9 : 29 + 64 = 93, lstm(29,64) = 115
```

可见差异较大，将样本增加到 10000 时，结果如下

num0 : 93 + 116 = 209, lstm(93,116) = 209
num1 : 114 + 37 = 151, lstm(114,37) = 151
num2 : 86 + 56 = 142, lstm(86,56) = 142
num3 : 49 + 96 = 145, lstm(49,96) = 145
num4 : 29 + 19 = 48, lstm(29,19) = 48
num5 : 100 + 22 = 122, lstm(100,22) = 122
num6 : 116 + 55 = 171, lstm(116,55) = 171
num7 : 124 + 2 = 126, lstm(124,2) = 126
num8 : 57 + 17 = 74, lstm(57,17) = 74
num9 : 30 + 82 = 112, lstm(30,82) = 112
num10 : 54 + 21 = 75, lstm(54,21) = 75
num11 : 4 + 55 = 59, lstm(4,55) = 59
num12 : 125 + 55 = 180, lstm(125,55) = 180
num13 : 9 + 74 = 83, lstm(9,74) = 83
num14 : 91 + 59 = 150, lstm(91,59) = 150
num15 : 66 + 114 = 180, lstm(66,114) = 180
num16 : 41 + 66 = 107, lstm(41,66) = 107
num17 : 122 + 81 = 203, lstm(122,81) = 203
num18 : 51 + 84 = 135, lstm(51,84) = 135
num19 : 111 + 57 = 168, lstm(111,57) = 168
num20 : 37 + 39 = 76, lstm(37,39) = 76
num21 : 122 + 18 = 140, lstm(122,18) = 140
num22 : 36 + 48 = 84, lstm(36,48) = 84
num23 : 107 + 90 = 197, lstm(107,90) = 197
num24 : 101 + 126 = 227, lstm(101,126) = 227
num25 : 28 + 117 = 145, lstm(28,117) = 145
num26 : 32 + 22 = 54, lstm(32,22) = 54

num27 : 42 + 26 = 68, lstm(42,26) = 68
num28 : 84 + 99 = 183, lstm(84,99) = 183
num29 : 119 + 127 = 246, lstm(119,127) = 246
num30 : 55 + 12 = 67, lstm(55,12) = 67
num31 : 7 + 70 = 77, lstm(7,70) = 77
num32 : 86 + 8 = 94, lstm(86,8) = 94
num33 : 72 + 103 = 175, lstm(72,103) = 175
num34 : 42 + 26 = 68, lstm(42,26) = 68
num35 : 111 + 76 = 187, lstm(111,76) = 187
num36 : 65 + 23 = 88, lstm(65,23) = 88
num37 : 70 + 35 = 105, lstm(70,35) = 105
num38 : 38 + 63 = 101, lstm(38,63) = 101
num39 : 34 + 26 = 60, lstm(34,26) = 60
num40 : 11 + 13 = 24, lstm(11,13) = 24
num41 : 80 + 74 = 154, lstm(80,74) = 154
num42 : 117 + 73 = 190, lstm(117,73) = 190
num43 : 67 + 29 = 96, lstm(67,29) = 96
num44 : 34 + 74 = 108, lstm(34,74) = 108
num45 : 108 + 10 = 118, lstm(108,10) = 118
num46 : 82 + 120 = 202, lstm(82,120) = 202
num47 : 65 + 52 = 117, lstm(65,52) = 117
num48 : 30 + 57 = 87, lstm(30,57) = 87
num49 : 108 + 77 = 185, lstm(108,77) = 185
num50 : 32 + 105 = 137, lstm(32,105) = 137
num51 : 16 + 123 = 139, lstm(16,123) = 139
num52 : 47 + 109 = 156, lstm(47,109) = 156
num53 : 50 + 70 = 120, lstm(50,70) = 120
num54 : 55 + 15 = 70, lstm(55,15) = 70
num55 : 37 + 7 = 44, lstm(37,7) = 44
num56 : 121 + 99 = 220, lstm(121,99) = 220
num57 : 99 + 35 = 134, lstm(99,35) = 134
num58 : 86 + 16 = 102, lstm(86,16) = 102
num59 : 97 + 59 = 156, lstm(97,59) = 156
num60 : 98 + 105 = 203, lstm(98,105) = 203
num61 : 10 + 73 = 83, lstm(10,73) = 83
num62 : 3 + 111 = 114, lstm(3,111) = 114
num63 : 117 + 7 = 124, lstm(117,7) = 124
num64 : 25 + 46 = 71, lstm(25,46) = 71
num65 : 56 + 37 = 93, lstm(56,37) = 93
num66 : 103 + 76 = 179, lstm(103,76) = 179
num67 : 48 + 123 = 171, lstm(48,123) = 171
num68 : 43 + 119 = 162, lstm(43,119) = 162
num69 : 1 + 13 = 14, lstm(1,13) = 14
num70 : 22 + 85 = 107, lstm(22,85) = 107

num71 : 124 + 126 = 250, lstm(124,126) = 250
num72 : 38 + 17 = 55, lstm(38,17) = 55
num73 : 105 + 113 = 218, lstm(105,113) = 218
num74 : 10 + 15 = 25, lstm(10,15) = 25
num75 : 87 + 52 = 139, lstm(87,52) = 139
num76 : 76 + 68 = 144, lstm(76,68) = 144
num77 : 103 + 37 = 140, lstm(103,37) = 140
num78 : 85 + 60 = 145, lstm(85,60) = 145
num79 : 108 + 86 = 194, lstm(108,86) = 194
num80 : 8 + 64 = 72, lstm(8,64) = 72
num81 : 99 + 103 = 202, lstm(99,103) = 202
num82 : 100 + 95 = 195, lstm(100,95) = 195
num83 : 45 + 5 = 50, lstm(45,5) = 50
num84 : 56 + 47 = 103, lstm(56,47) = 103
num85 : 12 + 122 = 134, lstm(12,122) = 134
num86 : 56 + 123 = 179, lstm(56,123) = 179
num87 : 40 + 34 = 74, lstm(40,34) = 74
num88 : 0 + 116 = 116, lstm(0,116) = 116
num89 : 123 + 102 = 225, lstm(123,102) = 225
num90 : 76 + 122 = 198, lstm(76,122) = 198
num91 : 50 + 2 = 52, lstm(50,2) = 52
num92 : 74 + 45 = 119, lstm(74,45) = 119
num93 : 103 + 120 = 223, lstm(103,120) = 223
num94 : 28 + 96 = 124, lstm(28,96) = 124
num95 : 74 + 14 = 88, lstm(74,14) = 88
num96 : 49 + 2 = 51, lstm(49,2) = 51
num97 : 43 + 47 = 90, lstm(43,47) = 90
num98 : 23 + 51 = 74, lstm(23,51) = 74
num99 : 37 + 10 = 47, lstm(37,10) = 47

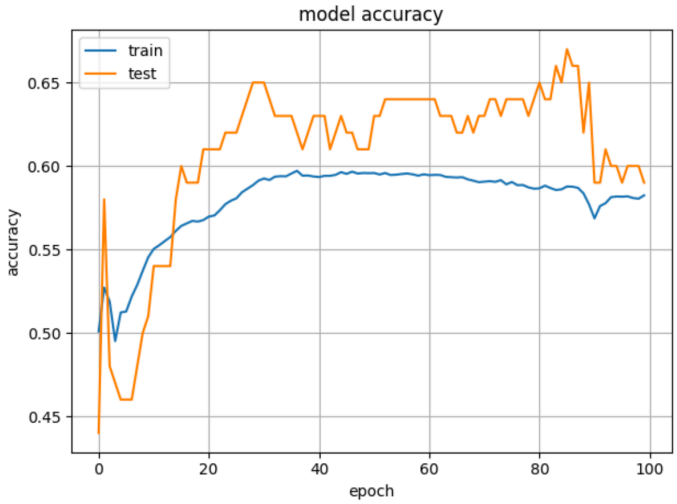
但当样本增加至 20000 时，准确度降低了，考虑可能是欠拟合。

五.测试结果：

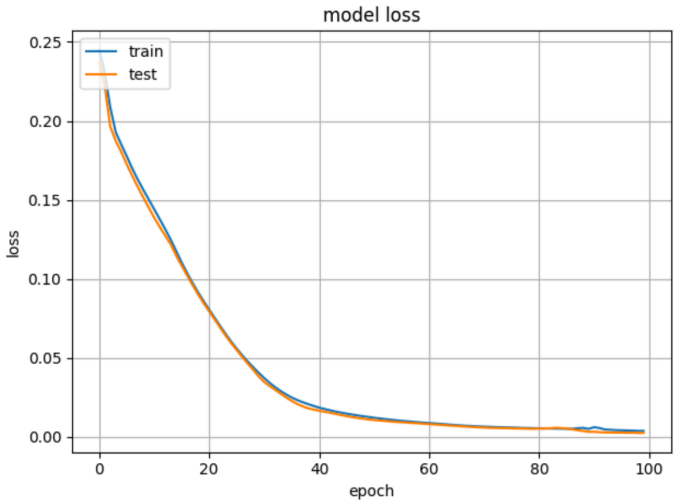
选择样本 10000，epoch100，batch128 为最好的模型进行训练

最终得到的测试集与训练集的 loss 与 accuracy 曲线如下

accuracy 曲线对比如下



loss 曲线对比如下



六.附录：

附录代码如下 文件为 lstm.py 以及 lstm0.py

lstm.py

```
    from keras.models import Sequential
from keras.layers import Dense, LSTM
from keras.utils import plot_model
import random
import numpy as np
import matplotlib.pyplot as plt

n = 10000
test_n = 100
```

```
def con2bin(x):
    x_list = []
    sx = '{:08b}'.format(x)
    for bi in sx:
        x_list.append(int(bi))
    return x_list

def bin2ten(x):
    weight = [2**7,2**6,2**5,2**4,2**3,2**2,2**1,2**0]
    index = np.where(x>0.5)
    new_x = np.zeros_like(x)
    new_x[index] = 1
    sum = new_x.dot(weight)
    return int(sum)

def get_data(n):
    data_x = []
    data_y = []
    for i in range(n):
        x1 = random.randint(0, 127)
        x2 = random.randint(0, 127)
        y = x1 + x2
        cx1 = con2bin(x1)
        cx2 = con2bin(x2)
        cy = con2bin(y)
        print(cy)
        data_x.append([cx1, cx2])
        data_y.append(cy)

    data_x = np.array(data_x)
    data_y = np.array(data_y)

    return data_x, data_y

data_x, data_y = get_data(n)
test_x, test_y = get_data(test_n)

print(data_x.shape)
print(data_y.shape)
print(test_x.shape)
print(test_y.shape)

model = Sequential()
```

```
model.add(LSTM(256, activation='relu',input_shape=(2, 8)))
model.add(Dense(8, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam',
metrics=['accuracy'])
LSTM = model.fit(data_x, data_y, epochs=100, batch_size=128,
validation_data=(test_x, test_y), verbose=2, shuffle=False)

plt.plot(LSTM.history['loss'])
plt.plot(LSTM.history['val_loss'])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train","test"],loc="upper left")
plt.grid()
plt.show()

plt.plot(LSTM.history['acc'])
plt.plot(LSTM.history['val_acc'])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.legend(["train","test"],loc="upper left")
plt.grid()
plt.show()

plot_model(model, to_file=r'./model.png', show_shapes=True)

for i in range(test_n):
    x1 = random.randint(0, 128)
    x2 = random.randint(0, 128)
    y = x1 + x2
    cx1 = con2bin(x1)
    cx2 = con2bin(x2)
    cy = con2bin(y)
    input_x = np.array([cx1,cx2])
    pre = model.predict(input_x.reshape(1,2,8))
    pre = bin2ten(pre.reshape(-1))
    print('num{} : {} + {} = {}, lstm({},{}) =
{}'.format(i,x1,x2,y,x1,x2,pre))

model.save('lstm_model.h5')
```

lstm0.py

```python
import numpy as np

# 输出单元激活函数
def softmax(x):
    x = np.array(x)
    max_x = np.max(x)
    return np.exp(x - max_x) / np.sum(np.exp(x - max_x))

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

def tanh(x):
    return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))


class LSTM:
    def __init__(self, data_dim, hidden_dim=100):
        # data_dim: 词向量维度，即词典长度; hidden_dim: 隐单元维度
        self.data_dim = data_dim
        self.hidden_dim = hidden_dim

        # 初始化权重向量
        self.whi, self.wxi, self.bi = self._init_wh_wx()
        self.whf, self.wxf, self.bf = self._init_wh_wx()
        self.who, self.wxo, self.bo = self._init_wh_wx()
        self.wha, self.wxa, self.ba = self._init_wh_wx()
        self.wy, self.by = np.random.uniform(-np.sqrt(1.0 / self.hidden_dim),
np.sqrt(1.0 / self.hidden_dim),
                                             (self.data_dim, self.hidden_dim)), \
                           np.random.uniform(-np.sqrt(1.0 / self.hidden_dim),
np.sqrt(1.0 / self.hidden_dim),
                                             (self.data_dim, 1))

    # 初始化 wh, wx, b
    def _init_wh_wx(self):
        wh = np.random.uniform(-np.sqrt(1.0 / self.hidden_dim), np.sqrt(1.0 /
self.hidden_dim),
                               (self.hidden_dim, self.hidden_dim))
        wx = np.random.uniform(-np.sqrt(1.0 / self.data_dim), np.sqrt(1.0 /
self.data_dim),
                               (self.hidden_dim, self.data_dim))
        b = np.random.uniform(-np.sqrt(1.0 / self.data_dim), np.sqrt(1.0 /
self.data_dim),
                              (self.hidden_dim, 1))
```

```
        return wh, wx, b

    # 初始化各个状态向量
    def _init_s(self, T):
        iss = np.array([np.zeros((self.hidden_dim, 1))] * (T + 1))  # input
gate
        fss = np.array([np.zeros((self.hidden_dim, 1))] * (T + 1))  # forget
gate
        oss = np.array([np.zeros((self.hidden_dim, 1))] * (T + 1))  # output
gate
        ass = np.array([np.zeros((self.hidden_dim, 1))] * (T + 1))  # current
inputstate
        hss = np.array([np.zeros((self.hidden_dim, 1))] * (T + 1))  # hidden
state
        css = np.array([np.zeros((self.hidden_dim, 1))] * (T + 1))  # cell
state
        ys = np.array([np.zeros((self.data_dim, 1))] * T)  # output value

        return {'iss': iss, 'fss': fss, 'oss': oss,
                'ass': ass, 'hss': hss, 'css': css,
                'ys': ys}

    # 前向传播，单个x
    def forward(self, x):
        # 向量时间长度
        T = len(x)
        # 初始化各个状态向量
        stats = self._init_s(T)

        for t in range(T):
            # 前一时刻隐藏状态
            ht_pre = np.array(stats['hss'][t - 1]).reshape(-1, 1)

            # input gate
            stats['iss'][t] = self._cal_gate(self.whi, self.wxi, self.bi,
ht_pre, x[t], sigmoid)
            # forget gate
            stats['fss'][t] = self._cal_gate(self.whf, self.wxf, self.bf,
ht_pre, x[t], sigmoid)
            # output gate
            stats['oss'][t] = self._cal_gate(self.who, self.wxo, self.bo,
ht_pre, x[t], sigmoid)
            # current inputstate
```

```
            stats['ass'][t] = self._cal_gate(self.wha, self.wxa, self.ba,
ht_pre, x[t], tanh)

            # cell state, ct = ft * ct_pre + it * at
            stats['css'][t] = stats['fss'][t] * stats['css'][t - 1] +
stats['iss'][t] * stats['ass'][t]
            # hidden state, ht = ot * tanh(ct)
            stats['hss'][t] = stats['oss'][t] * tanh(stats['css'][t])

            # output value, yt = softmax(self.wy.dot(ht) + self.by)
            stats['ys'][t] = softmax(self.wy.dot(stats['hss'][t]) + self.by)

        return stats

    # 计算各个门的输出
    def _cal_gate(self, wh, wx, b, ht_pre, x, activation):
        return activation(wh.dot(ht_pre) + wx[:, x].reshape(-1, 1) + b)

    # 预测输出，单个x
    def predict(self, x):
        stats = self.forward(x)
        pre_y = np.argmax(stats['ys'].reshape(len(x), -1), axis=1)
        return pre_y

    # 计算损失， softmax 交叉熵损失函数， (x,y)为多个样本
    def loss(self, x, y):
        cost = 0
        for i in range(len(y)):
            stats = self.forward(x[i])
            # 取出 y[i] 中每一时刻对应的预测值
            pre_yi = stats['ys'][range(len(y[i])), y[i]]
            cost -= np.sum(np.log(pre_yi))

        # 统计所有 y 中词的个数，计算平均损失
        N = np.sum([len(yi) for yi in y])
        ave_loss = cost / N

        return ave_loss

    # 初始化偏导数 dwh, dwx, db
    def _init_wh_wx_grad(self):
        dwh = np.zeros(self.whi.shape)
        dwx = np.zeros(self.wxi.shape)
        db = np.zeros(self.bi.shape)
```

```
        return dwh, dwx, db

    # 求梯度，(x,y)为一个样本
    def bptt(self, x, y):
        dwhi, dwxi, dbi = self._init_wh_wx_grad()
        dwhf, dwxf, dbf = self._init_wh_wx_grad()
        dwho, dwxo, dbo = self._init_wh_wx_grad()
        dwha, dwxa, dba = self._init_wh_wx_grad()
        dwy, dby = np.zeros(self.wy.shape), np.zeros(self.by.shape)

        # 初始化 delta_ct，因为后向传播过程中，此值需要累加
        delta_ct = np.zeros((self.hidden_dim, 1))

        # 前向计算
        stats = self.forward(x)
        # 目标函数对输出 y 的偏导数
        delta_o = stats['ys']
        delta_o[np.arange(len(y)), y] -= 1

        for t in np.arange(len(y))[::-1]:
            # 输出层 wy, by 的偏导数，由于所有时刻的输出共享输出权值矩阵，故所有时刻累加
            dwy += delta_o[t].dot(stats['hss'][t].reshape(1, -1))
            dby += delta_o[t]

            # 目标函数对隐藏状态的偏导数
            delta_ht = self.wy.T.dot(delta_o[t])

            # 各个门及状态单元的偏导数
            delta_ot = delta_ht * tanh(stats['css'][t])
            delta_ct += delta_ht * stats['oss'][t] * (1 -
tanh(stats['css'][t]) ** 2)
            delta_it = delta_ct * stats['ass'][t]
            delta_ft = delta_ct * stats['css'][t - 1]
            delta_at = delta_ct * stats['iss'][t]

            delta_at_net = delta_at * (1 - stats['ass'][t] ** 2)
            delta_it_net = delta_it * stats['iss'][t] * (1 - stats['iss'][t])
            delta_ft_net = delta_ft * stats['fss'][t] * (1 - stats['fss'][t])
            delta_ot_net = delta_ot * stats['oss'][t] * (1 - stats['oss'][t])

            # 更新各权重矩阵的偏导数，由于所有时刻共享权值，故所有时刻累加
            dwhf, dwxf, dbf = self._cal_grad_delta(dwhf, dwxf, dbf,
delta_ft_net, stats['hss'][t - 1], x[t])
```

```
            dwhi, dwxi, dbi = self._cal_grad_delta(dwhi, dwxi, dbi,
delta_it_net, stats['hss'][t - 1], x[t])
            dwha, dwxa, dba = self._cal_grad_delta(dwha, dwxa, dba,
delta_at_net, stats['hss'][t - 1], x[t])
            dwho, dwxo, dbo = self._cal_grad_delta(dwho, dwxo, dbo,
delta_ot_net, stats['hss'][t - 1], x[t])

        return [dwhf, dwxf, dbf,
                dwhi, dwxi, dbi,
                dwha, dwxa, dba,
                dwho, dwxo, dbo,
                dwy, dby]

    # 更新各权重矩阵的偏导数
    def _cal_grad_delta(self, dwh, dwx, db, delta_net, ht_pre, x):
        dwh += delta_net * ht_pre
        dwx += delta_net * x
        db += delta_net

        return dwh, dwx, db

    # 计算梯度, (x,y)为一个样本
    def sgd_step(self, x, y, learning_rate):
        dwhf, dwxf, dbf, \
        dwhi, dwxi, dbi, \
        dwha, dwxa, dba, \
        dwho, dwxo, dbo, \
        dwy, dby = self.bptt(x, y)

        # 更新权重矩阵
        self.whf, self.wxf, self.bf = self._update_wh_wx(learning_rate,
self.whf, self.wxf, self.bf, dwhf, dwxf, dbf)
        self.whi, self.wxi, self.bi = self._update_wh_wx(learning_rate,
self.whi, self.wxi, self.bi, dwhi, dwxi, dbi)
        self.wha, self.wxa, self.ba = self._update_wh_wx(learning_rate,
self.wha, self.wxa, self.ba, dwha, dwxa, dba)
        self.who, self.wxo, self.bo = self._update_wh_wx(learning_rate,
self.who, self.wxo, self.bo, dwho, dwxo, dbo)

        self.wy, self.by = self.wy - learning_rate * dwy, self.by -
learning_rate * dby

    # 更新权重矩阵
    def _update_wh_wx(self, learning_rate, wh, wx, b, dwh, dwx, db):
```

```
        wh -= learning_rate * dwh
        wx -= learning_rate * dwx
        b -= learning_rate * db

        return wh, wx, b

    # 训练 LSTM
    def train(self, X_train, y_train, learning_rate=0.005, n_epoch=5):
        losses = []
        num_examples = 0

        for epoch in range(n_epoch):
            for i in range(len(y_train)):
                self.sgd_step(X_train[i], y_train[i], learning_rate)
                num_examples += 1

            loss = self.loss(X_train, y_train)
            losses.append(loss)
            print('epoch {0}: loss = {1}'.format(epoch + 1, loss))
            if len(losses) > 1 and losses[-1] > losses[-2]:
                learning_rate *= 0.5
                print('decrease learning_rate to', learning_rate)

lstm = LSTM(128)
n = 1000
data_x = np.random.randint(64,size=(n,2))
data_y = np.sum(data_x, axis=1).reshape(n,1)
lstm.train(data_x, data_y)

test_n = 10
for i in range(test_n):
    x = np.random.randint(64,size=(2))
    y = lstm.predict(x)
    print(x)
    print(y)
```