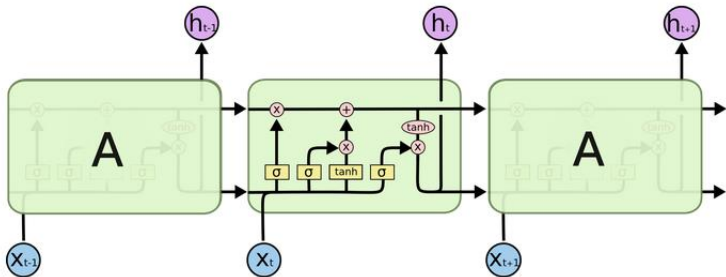
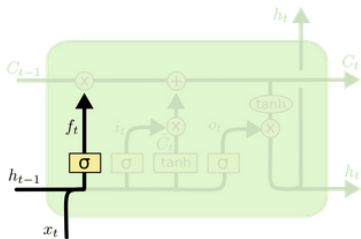
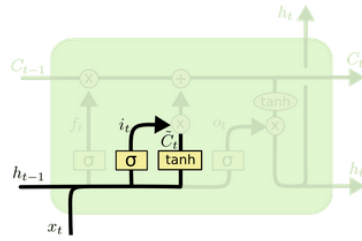


认知科学与类脑计算课程实验报告

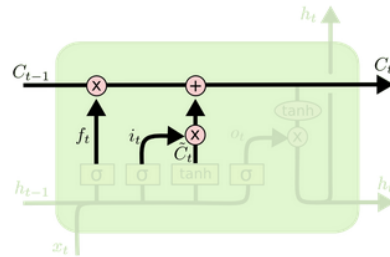
实验题目：LSTM 的实现		学号：201600181073
日期：2019. 5. 24	班级：智能 16	姓名：唐超
实验目的： 使用 Python 语言实现基于 LSTM 模型的八位二进制整数（128 以内）的加法运算。		
实验环境： Jupyter notebook & python3.6 & numpy		
主要算法及步骤： <p>LSTM 是一类可以处理长期依赖问题的特殊的 RNN，LSTM 主要用来处理长期依赖问题，与传统 RNN 相比，长时间的信息记忆能力是与生俱来的。所有的 RNN 链式结构中都有不断重复的模块，用来随时间传递信息。</p> <p>LSTM 链式结构中重复模块的结构有四个互相交互的层 (如下图所示)。</p>  <p>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。</p> <p>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。</p>  $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ <p>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。</p>		



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

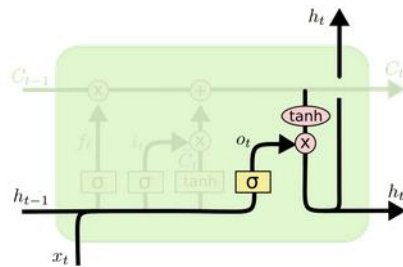
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

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。



$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

程序设计：

1. 二进制整数加法运算

定义一个二进制整数加法运算函数，其输入输出均为二进制数字字符串。

```
# 二进制加法, 输入为两个0, 1字符串
def binaryAddition(a, b):
    h = 0
    for i in range(len(a)):
        h += 2**(len(a)-1-i)*int(a[i])
    for i in range(len(b)):
        h += 2**(len(b)-1-i)*int(b[i])
    return '{:08b}'.format(h)
```

2. 组织训练数据：

生成 0-127 之间的两个八位二进制整数的所有组合，将每组两个数的和（用 1 中 binaryAddition 函数计算）作为其标签，即为训练数据，分别存入 numpy 数组中。由于创建网络的需要，将 `x_train` 的形状 reshape 为 (128*128, 1, 8*2)，`y_train` 的形状仍为 (128*128, 8)，具体代码如下：

```

bNumStr = []
for i in range(128):
    # 将0~128转换为二进制字符串并保存到列表中
    bNumStr.append(' {:08b}'.format(i))

x_train = np.zeros((128*128, 16))
y_train = np.zeros((128*128, 8))
c = 0
for i in range(128):
    for j in range(128):
        group = bNumStr[i]+bNumStr[j]
        for k in range(len(group)):
            x_train[c][k] = int(group[k])

        y = binaryAddition(bNumStr[i], bNumStr[j])
        for k in range(len(y)):
            y_train[c][k] = int(y[k])
        c += 1

x_train = x_train.reshape(128*128, 1, 16)

```

3. 训练网络

这一部分用 keras 实现，设置 LSTM 层神经元个数为 256，输入序列形状为 (1, 16)，在 LSTM 层后加一个全连接层，输出向量维度为 8，激活函数为 sigmoid。训练时，将均方误差作为 loss 函数，adam 作为优化器，设置 batch_size 为 128，训练 250 个 epochs，最终 accuracy=0.4854。

```

#build model
model = Sequential()
model.add(LSTM(units=256, input_shape=(1, 16)))
model.add(Dense(8, activation='sigmoid'))

#compile:loss, optimizer, metrics
model.compile(loss=losses.mean_squared_error, optimizer='adam', metrics=['accuracy'])

#train: epoch, batch_size
model.fit(x_train, y_train, epochs=250, batch_size=128, verbose=1)

```

4. 测试

定义测试函数 predict，输入为待测试的两个八位二进制整数字符串，同时用 binaryAddition 函数和上一步训练的 LSTM 网络计算和，当两个计算结果相等时，输出 True，否则输出 False。

```

# 测试 输入a, b是两个八位二进制数字字符串
def predict(a, b):
    print(a+' '+b)
    result1 = binaryAddition(a, b)
    print("二进制加法计算结果: \n\tresult1 = ", result1)
    add = np.zeros((1, 1, 16))
    for i in range(8):
        add[0][0][i] = int(a[i])
        add[0][0][i+8] = int(b[i])
    r = model.predict(add)[0]
    r[r>=0.5] = 1
    r[r<0.5] = 0
    result2 = ''
    for i in range(8):
        result2 += str(int(r[i]))
    print("LSTM计算结果: \n\tresult2 = ", result2)
    print("result1==result2 ? ", result1==result2)
    return result1==result2

```

调试分析：

只用 LSTM 层，设置 8 个神经元，输出时默认不使用激活函数，loss 函数作为交叉熵损失函数，最终 accuracy 为 0.5 左右。

经过调试与分析，最后设置 256 个神经元，加入 sigmoid 作为激活函数的全连接层，均方误差作为损失函数，改进后的 accuracy 达到 0.6 左右。

输入数据经过整个网络复杂的计算，输出结果很难达到与 label 完全一致，分析此处的 accuracy 的计算过程，当输出向量的某一个元素的值与 label 中该位置元素十分接近时，由于舍入误差的存在，计算机将其视为相等。通过实验验证：

```
In [117]: p = np.zeros((1,2))
          p[0] = 0.9999999999999995
          q = np.zeros((1,2))
          q[0] = 1
          p == q

Out[117]: array([[ True,  True]])
```

因此，一方面，这里的 accuracy 不能完全反映模型效果，另一方面，受此启发，可以通过对网络输出结果设置阈值提高准确度。设置阈值为 0.5，大于 0.5 的置为 1，否则置为 0，此时对所有训练数据进行测试，准确率达到 100%。

测试结果及分析：

部分测试结果如下：

```
01011001 + 00111001
二进制加法计算结果：
      result1 = 10010010
LSTM计算结果：
      result2 = 10010010
result1==result2 ? True
```

```
01011011 + 00100001
二进制加法计算结果：
      result1 = 01111100
LSTM计算结果：
      result2 = 01111100
result1==result2 ? True
```

经测试，该网络对所有八位二进制数（128 以内）的加法准确率达到 100%。