第7章 深度学习

班级： 无61 姓名： 李溢 学号： 2016011235

1. **神经网络BP算法**

考虑一个神经元，它采用 Sigmoid函数作为激活函数。神经元输入为， 为输入数据的个数，偏置量；权值向量为； ，神经元输出为，样本对应的类别真值为；误差函数为。

1. 写出上述误差函数对权重向量的雅克比Jacobian矩阵形式；在此基础上写出权值更新公式（学习率为）；

注：Jacobian矩阵是一阶偏导数以一定形式排列成的矩阵；假设有函数则的Jacobian矩阵定义为，即Jacobian矩阵的第行第列的元素是函数输出的第个元素对第个输入的导数。

1. 神经元输入为，样本对应的类别真值，权值，即，设学习率，请计算误差反向传播之后、权值更新一次得到的数值。

（1）一些偏导数如下：

根据链式法则有：

因此 Jacobian 矩阵形式为

权重更新公式为：

（2）

即

**2. (上机题) 卷积神经网络CNN**

**本题责任助教：曾子愚 zy-zeng18@mails.tsinghua.edu.cn**

利用numpy实现基于MNIST手写数字数据集上卷积神经网络的训练和测试

1. 填写mnist\_train.py中的第一部分，在代码中的COMPLETE处填入你的代码，实现MNIST数据集的装载(注：使用mnist\_train.py中的load()函数从mnist.pkl中读入train\_images, train\_labels, test\_images, test\_labels 4个变量；并可使用show()函数显示训练集中第一张图片，并检查其对应标签。)， 需完善的代码包括：
2. 归一化函数normalize()，该函数输入是图片集合（如训练集X\_train），输出是归一化后的图片集合，使用归一化函数分别对训练集和测试集图片进行归一化，使得CNN输入取值归一化到[-1,1]区间。
3. 标签转化函数label2OH()和OH2label()，其中函数label2OH()将数据集中标签转化为one-hot编码；函数OH2label()，将one-hot编码转化为标签形式，对比转化后的标签与转化前的标签是否一致。

注：one-hot编码含义是假设总共有C个类别，标签为k的one-hot编码为二进制向量，其中仅第k位为1，其余位为0；

（1）a)归一化

x = (x-x.min()) / x.max()\*2 - 1

b)标签转化函数

def label2OH(y,D\_out):

N = y.shape[0]

OH = np.zeros((N,D\_out))

OH[np.arange(N), y] = 1

return OH

def OH2label(OH):

y = np.argmax(OH, axis=1)

return y

1. 完成mnist\_train.py中的第二部分，在代码中的COMPLETE处填入你的代码，完成ReLU、Softmax、FC层的前向与反向传播。
   1. 对于隐含层，我们采用ReLU作为激活函数，结合ReLU激活函数定义和导数形式，完成ReLU的前向传播和反向传播；

注：假设已知正向传播函数的导数Jacobian矩阵，并给定反向传播的上一步输入，则其反向传播输出计算公式为。对于ReLU这种逐元素的运算，则反向传播的输出只需要是和的逐元素相乘。

* 1. 对于输出层，我们采用Softmax作为输出单元。在MNIST手写数字识别任务中，我们将输入图像分为10类。给定输入，Softmax输出，其中每一维代表输入图像属于第*i*类的概率。结合Softmax激活函数的数学表达，实现Softmax的正向传播和反向传播；

注1：实际实现Softmax时，由于用到指数函数，我们需要限制softmax输入的大小，一个通用简单的方法是减去每行的最大值：

基于这个公式，完成softmax的正向传播，注意实际网络中softmax的输入维度是[batch\_size,10]；

注2：Softmax的导数为

* 1. 全连接FC层的正向传播表达式为，其中输入， 推导FC层的导数，并完成FC层的正向传播和反向传播。

（2）a)ReLU

def \_forward(self, x):

out = np.maximum(0,x)

self.input = x

return out

def \_backward(self, d):

dX = d\*((self.input)>0)

return dX

b)Softmax

def \_forward(self, X):

Y = np.exp(X - np.tile(X.max(axis=1).reshape([X.shape[0],1]), X.shape[1]))

Z = Y / np.tile(np.sum(Y, axis=1).reshape([Y.shape[0],1]), Y.shape[1])

self.input = X

self.output = Z

return Z # distribution

def \_backward(self, dout):

X = self.input

Z = self.output

dX = np.zeros(X.shape)

N = Z.shape[0]

for n in range(N):

J = - np.dot(Z[n,:].reshape([Z.shape[1],1]), Z[n,:].reshape([1,Z.shape[1]])) + np.diag(Z[n,:])

dX[n,:] = np.dot(J,dout[n,:])

return dX

c)

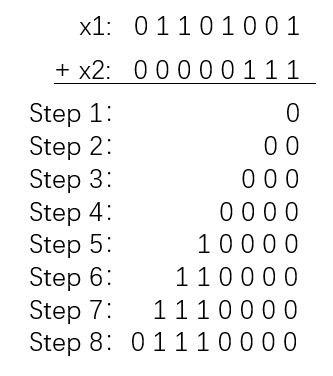
1. 训练模型，运行mnist\_train.py，即可在mnist上训练一个简单的两层神经网络。尝试修改训练epoch数、初始学习率lr、batch\_size、隐含单元维度H、网络层数等，观察其对训练集和验证集上结果的影响。
2. (选做题)尝试更多模块如卷积层、Dropout、Pooling等的实现，并采用更复杂的网络如LeNet、ResNet等，比较识别准确率。

**3. (上机题) 循环神经网络**

**本题责任助教：郑悦 y-zheng13@mails.tsinghua.edu.cn**

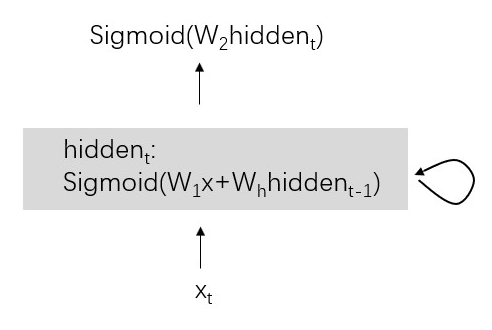
利用numpy实现一个简单的RNN网络，并利用该网络实现二进制加法的功能。

对于两个输入的二进制数，比如 1101001 和 111 ，我们可以直接计算得到它们的和 1110000 。另一方面，二进制数也可以被视为一个序列，那么这个过 程可以看作是处理两个序列并得到一个序列的过程。假定我们有一个模型， 它可以对输入的两个 0,1 序列逐个位置进行处理，并输出相应位置的求和结 果，那么我们可以逐个数字得到整个序列的求和结果。 显然在处理每一个位置时，这个模型不仅需要考虑当前位置的两个输入，还 需要考虑先前的计算结果（是否有进位）



本题我们用一个RNN网络来实现模型的功能，结合代码 rnn\_problem.py 完成练习。

1. 准备训练数据集并实现Sigmoid函数。在Q1部分实现Sigmoid运算，并推导Sigmoid函数的导数，在Q2 部分实现相应的运算，运行并检查实现是否正确;
2. 设置模型参数，进行初始化，并设置模型的学习率lr。其中模型的输入x 是dim=2的vector，分别表示进行求和的两个二进制数的其中一位；模型输出是dim=1的vector，表示相应位上的求和结果。使用的rnn模型包含一个隐含层 hidden layer，各个layer之间的连接结构为



根据上述模型推导并在W\_grad 中储存相应的导数；

1. 训练RNN模型。在每一次训练中，我们使用RNN进行一次正向传播，得到模型的预测结果 pred 和 loss 。整个序列的正向计算结束后，利用反向传播计算出变量和模型参数的导数。请结合要求的RNN结构实现 Q3 , Q4；根据Q4得到的结果，使用交叉熵损失函数Cross-Entropy loss = - (ylog(p) + (1-y)log(1-p)) 并保存于 loss；进一步在Q5 根据 Q4 中得到的输出判断输出的 0,1 运算结果；在Q6 部分，需要实现各个参数的求导，首先计算各个参数 ∂loss/∂W 的解析表达以及变量 ∂loss/∂hidden\_t 会在不同位的计算中传递，写出 ∂loss/∂hidden\_t 的表达式，并在代码实现导数的运算。

完成上述代码后，根据反向传播的结果更新模型参数，运行并观察模型的迭代过程。

注：迭代loss值参考



(4) (选做题) 试分析模型的学习率对收敛速度有怎样的影响以及隐含层hidden layer的维数对模型有怎样的影响？