1 理论题

1.1 单热向量与交叉熵损失函数

a) 假设:

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{Y} = \begin{pmatrix} \mathbf{y1} \\ \mathbf{y2} \\ \mathbf{y3} \end{pmatrix}$$

则有:

$$\begin{aligned} loss &= -\sum_{i,j} Z_{ij} * ln\frac{e^{y_{ij}}}{\sum_k e^{y_{ik}}} \\ &= -(ln\frac{e^{y_{11}}}{\sum_k e^{y_{1k}}} + ln\frac{e^{y_{22}}}{\sum_k e^{y_{2k}}} + ln\frac{e^{y_{33}}}{\sum_k e^{y_{3k}}}) \end{aligned} \tag{1}$$

b) 由等式（1），只需把 y1，y2，y3 代入即可得到

$$\begin{aligned} loss &= -(ln\frac{e^{0.65}}{\sum_k e^{y_{1k}}} + ln\frac{e^{0.51}}{\sum_k e^{y_{2k}}} + ln\frac{e^{0.72}}{\sum_k e^{y_{3k}}}) \\ &= -(ln0.42 + ln0.42 + ln0.44) \\ &= 2.5469 \end{aligned}$$

1.2 矩阵求导

先证明引理：对于一标量 y,m*n 的矩阵 X, 若有 $y = f(X)$, 则有:

$$d\mathbf{y} = tr\Big((\frac{\partial y}{\partial X})^T dX\Big) \tag{2}$$

由全微分的定义:

$$df = \sum_{i,j} \frac{\partial y}{\partial X_{ij}} dX_{ij}$$

而由矩阵的迹的性质,对于两个尺寸相同的矩阵 A,B,有 $tr(A^T B) = \sum_{i,j} A_{ij} B_{ij}$,
因此:

$$d\mathbf{y} = tr\Big((\frac{\partial y}{\partial X})^T dX\Big) \tag{3}$$

对于标量 J，损失函数 $\mathbf{J} = ||\mathbf{y} - \mathbf{Z}||$，可以写成 $\mathbf{J} = (\mathbf{y} - \mathbf{Z})^T (\mathbf{y} - \mathbf{Z})$
对 $\mathbf{J}$ 求全微分，有

$$d\mathbf{J} = 2(\mathbf{y} - \mathbf{Z})^T d\mathbf{y} \tag{4}$$

下面推导 $d\mathbf{y}$ 的表达式：
由 y 的表达式 $\mathbf{y} = sigmoid(\mathbf{W}^T\mathbf{x} + \mathbf{b})$，求导得：

$$\begin{aligned}
d\mathbf{y} &= \frac{e^{\mathbf{W}^T\mathbf{x}+\mathbf{b}}}{(1+e^{\mathbf{W}^T\mathbf{x}+\mathbf{b}})^2} d(\mathbf{W}^T\mathbf{x}+\mathbf{b}) \\
&= \frac{1}{1+e^{\mathbf{W}^T\mathbf{x}+\mathbf{b}}} \odot \left(1 - \frac{1}{1+e^{\mathbf{W}^T\mathbf{x}+\mathbf{b}}}\right) d(\mathbf{W}^T\mathbf{x}+\mathbf{b}) \\
&= \frac{1}{1+e^{\mathbf{W}^T\mathbf{x}+\mathbf{b}}} \odot \left(1 - \frac{1}{1+e^{\mathbf{W}^T\mathbf{x}+\mathbf{b}}}\right) (d\mathbf{W}^T\mathbf{x}+d\mathbf{b}) \\
&= \mathbf{y} \odot (1-\mathbf{y})(d\mathbf{W}^T\mathbf{x}+d\mathbf{b})
\end{aligned}$$

其中 $\odot$ 代表两个尺寸相同的矩阵各元素相乘 (element wise product)，下同
把 $d\mathbf{y}$ 代入（2）式中，可得全微分公式：

$$d\mathbf{J} = 2tr\Big(\big((\mathbf{y}-\mathbf{Z})\odot\mathbf{y}\odot(1-\mathbf{y})\big)\mathbf{x}^T d\mathbf{W} + \big((\mathbf{y}-\mathbf{Z})\odot\mathbf{y}\odot(1-\mathbf{y})\big)^T d\mathbf{b}\Big)$$

同时由全微分和矩阵的迹的关系：

$$d\mathbf{y} = tr\Big(\sum_i \big(\frac{\partial \mathbf{y}}{\partial \mathbf{x_i}}\big)^T d\mathbf{x_i}\Big) \tag{5}$$

可以推出：

$$\begin{aligned}
\frac{\partial \mathbf{J}}{\partial \mathbf{W}} &= 2\mathbf{x}\big((\mathbf{y}-\mathbf{Z})\odot\mathbf{y}\odot(1-\mathbf{y})\big)^T \\
\frac{\partial \mathbf{J}}{\partial \mathbf{b}} &= 2(\mathbf{y}-\mathbf{Z})\odot\mathbf{y}\odot(1-\mathbf{y})
\end{aligned}$$

2 代码题

2.1 iris

iris 数据集含有 150 个样本，其中数据部分含有 4 个特征，标签是 3 类的单热向量，因此神经网络输入层含有 4 个神经元，输出层含有 3 个神经元。隐含层的神经元数可以当作一个超参数 h。假设该神经网络的输入为一个（m，4）的矩阵，则输入层到隐含层的权重 **W1** 的尺寸为（4，h），偏移量 **b1** 的尺寸为（1，h）。同理，隐含层到输出层的权重 **W2** 的尺寸为（h，3），偏移量 **b2** 的尺寸为（1，3）。由于该网络是为解决多分类问题，输出层的激活函数使用 softmax 最佳，损失函数使用交叉熵函数。tensorflow 中的 tf.losses.softmax_cross_entropy 函数的表达式如下：

$$loss = -\sum_i y'_i * log(softmax(y)_i)$$

而隐含层的激活函数，我分别使用了 relu，sigmoid 和 tanh。以下为不同激活函数的准确率和损失函数变化曲线：

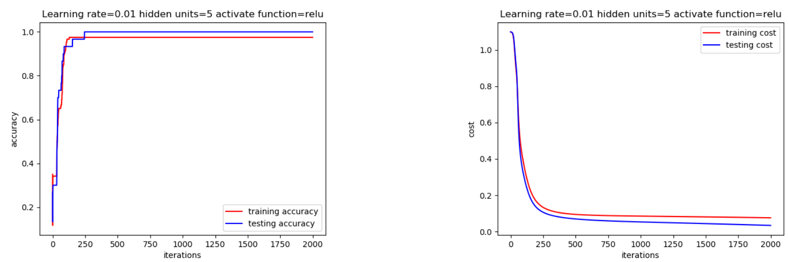


Figure 1: relu

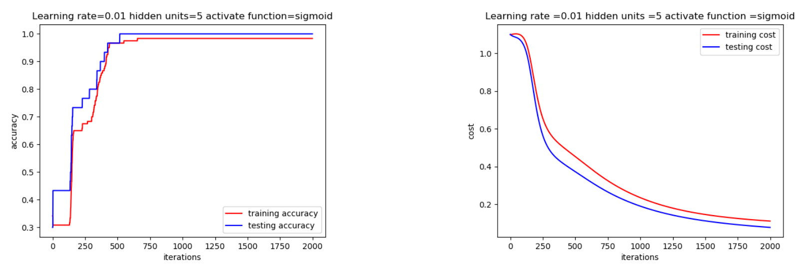


Figure 2: sigmoid

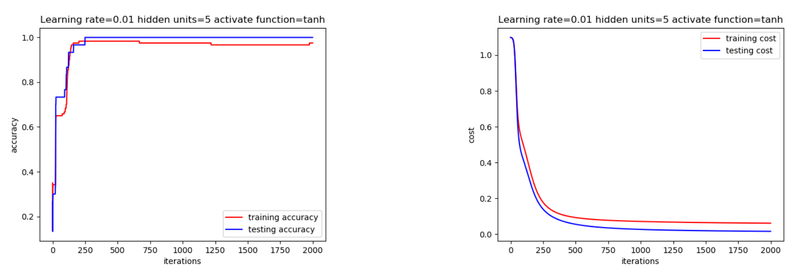
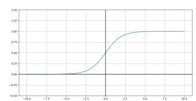


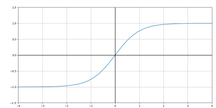
Figure 3: tanh

我们可以很明显的发现，使用 sigmoid 函数后的，损失函数的下降速率明显低于其他两个，而 relu 和 tanh 之间，relu 又略优于 tanh。在运行速度上（在同一机器），使用 relu 函数，每训练一轮，要经过 1.264ms，而 sigmoid 和 tanh 分别为 1.424ms 和 1.475ms，也就是说 relu 函数的运算速度要略快于其他两个函数。

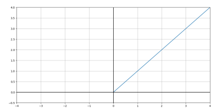
究其原因，我们可以看看每个函数的数学表达式和函数曲线：



$\mathbf{y} = \frac{1}{1+e^{-x}}$



$\mathbf{y} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



$\mathbf{y} = max(x, 0)$

从时间因素来看，tanh 和 sigmoid 函数中包含指数运算，因此不管是正向传播还是反向传播，都需要进行指数运算，而 relu 函数都是线性运算，所需的计算时间自然较少。撇开时间因素不谈，这三个函数还有以下的不同：

a) 对于 sigmoid 函数来说：

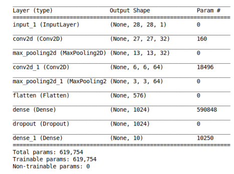
    1. 在输入很大或者很小的情况下，sigmoid 函数的导数值都几乎为 0. 而反向传播正需要梯度来更新参数，梯度越小，损失函数值自然下降得也越慢。因此如果在对参数进行初始化的时候使参数过大，那么该网络中大多数的神经元都出现了梯度消失的现象，那么损失值的下降将会很慢。
    2. sigmoid 函数的函数值是以 0.5 为中心的。这会导致后层的神经元的输入是非 0 均值的信号，这会对梯度产生影响：假设后层神经元的输入都为正, 那么对 w 求局部梯度则都为正，这样在反向传播的过程中 w 要么都往正方向更新，要么都往负方向更新，导致有一种捆绑的效果，从而导致梯度下降权重更新时出现 z 字型的下降，使得收敛缓慢。

b) tanh 函数解决了 sigmoid 函数值不以 0 为中心的问题，但是梯度消失的问题依然存在

c) relu 函数的线性和非饱和性很好的解决了梯度消失的问题。但他还有一个很明显的缺点，就是当一个很大的梯度经过一个神经元的时候，更新参数之后，导致该神经元以后的梯度都为 0，也就是说该神经元的参数停止更新，这将大幅度的降低神经网络的性能。因此学习率的设置就显得尤为重要，合理的学习率会降低这一情况发生的概率

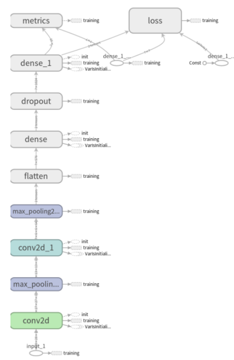
2.2 mnist

mnist 数据集的数据为 70000 张大小为 28*28 的手写数字图片。对于图像处理，使用卷积神经网络最为合适。以下构建 CNN 网络架构：



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d (Conv2D)	(None, 27, 27, 32)	160
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 6, 6, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 1024)	590848
dropout (Dropout)	(None, 1024)	0
dense_1 (Dense)	(None, 10)	10250

Total params: 619,754
Trainable params: 619,754
Non-trainable params: 0



- 输入层
- conv1，含有 32 个F积内核，内核大小为（2，2），步长为 1，输出大小为（None，27，27，32），激活函数为 relu
- maxpooling1，内核大小（2，2），步长为 2，输出大小为（None，13，13，32）
- conv2，含有 64 个F积内核，内核大小为（3，3），步长为 2，输出大小为（None，6，6，64），激活函数为 relu
- maxpooling2，内核大小（2，2），步长为 2，输出大小为（None，3，3，64）
- 全连接层，输出大小为（None，1024），dropout 概率为 0.6，激活函数为 relu
- 全连接层，输出大小为（None，10），激活函数为 softmax

优化方法使用 adam 优化器，学习率默认 0.0001，损失函数使用交叉熵, 训练 12 次测试集准确率大概能达到 99.25%, 训练过程中准确率和损失函数值变化情况如下图：

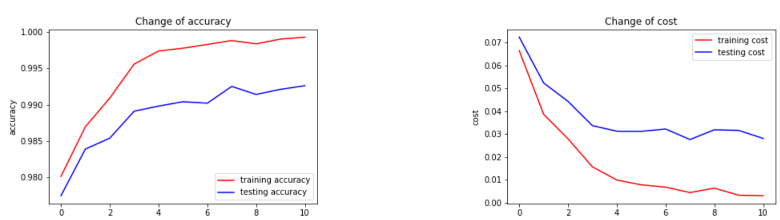


Figure 4: relu

2.3 附加题

假设预测值为 $\mathbf{y}$, 真实值为 $\mathbf{y'}$, 按要求可得，输入 $\mathbf{y}$ 与输出 $\mathbf{x}$ 的关系为：

$$\mathbf{y} = sigmoid(\mathbf{XW_1} + \mathbf{b_1})\mathbf{W_2} + \mathbf{b_2}$$

交叉熵损失函数表达式为：

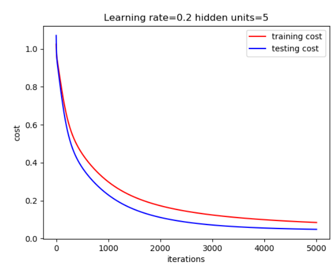
$$loss = -\sum_i y_i' * log(softmax(y)_i)$$

设 $Z_1 = XW_1 + b_1, A_1 = sigmoid(Z_1), Z_2 = A_1W_2 + b_2, A_2 = softmax(Z_2)$, 由此计算导数：

$$\frac{\partial loss}{\partial Z_2} = A_2 - y' \qquad \frac{\partial loss}{\partial W_2} = A_1^T \frac{\partial loss}{\partial Z_2} \qquad \frac{\partial loss}{\partial b_2} = \frac{\partial loss}{\partial Z_2}$$

$$\frac{\partial loss}{\partial Z_1} = \frac{\partial loss}{\partial Z_2} W_2^T \odot A1 \odot (1 - A1) \qquad \frac{\partial loss}{\partial W_1} = X^T \frac{\partial loss}{\partial Z_1} \qquad \frac{\partial loss}{\partial b_1} = \frac{\partial loss}{\partial Z_1}$$

由以上表达式在代码中构建正向传播和反向传播，完成对损失函数的优化。以下为优化过程中准确率和损失函数值的变化曲线图：



与使用 tensorflow 框架对比，在性能上来说区别不大，只是相应的最优学习率和训练次数有所差别，最终的结果差别不大。但明显的发现，使用 numpy 自行实现的程序的运行速度要比 tensorflow 快得多（前者 0.42ms 每步而后者 1.424ms）。我认为应该是原因在于自行实现的程序没有那么多的依赖库，因此节省了很多不必要的时间消耗。而使用 tensorflow 处理这一数据量较小的数据集，无异于杀鸡而用牛刀。

3 总结

本次作业主要考察的地方有：

- 交叉熵损失函数与 softmax 函数的配合使用计算损失
- 机器学习框架的使用
- 多层神经网络的构建与不同激活函数的使用

- 卷积神经网络的构建
- 矩阵求导以及正向传播、反向传播的构建

总的来说，本次作业涉及的内容较多，对编程与线性代数均有一定的要求。作业完成期间与同学进行了很多深入的交流，不同的思想之间碰撞出了不少的火花，收益匪浅。