计算机科学与技术学院神经网络与深度学习课程实验报告

实验题目：实验 1		学号：201900161098
日期：10.4	班级： 智能 19	姓名： 马田慧
Email：tianhuima01@gmail.com		

实验目的：
图像分类，knn，svm，softmax，3 层神经网络已经相应的向量化实现

实验软件和硬件环境：
Dell，jupyter notebook

实验原理和方法：
按照算法，填充相应的代码；查找预备知识：
Python 与 numpy 相关：

广播 Broadcasting

广播是一种强有力的机制，它让 Numpy 可以让不同大小的矩阵在一起进行数学计算。我们常常会有一个小的矩阵和一个大的矩阵，然后我们会需要用小的矩阵对大的矩阵做一些计算。

对两个数组使用广播机制要遵守下列规则：

1. 如果数组的秩不同，使用 1 来将秩较小的数组进行扩展，直到两个数组的尺寸的长度都一样。
2. 如果两个数组在某个维度上的长度是一样的，或者其中一个数组在该维度上长度为 1，那么我们就说这两个数组在该维度上是**相容**的。
3. 如果两个数组在所有维度上都是相容的，他们就能使用广播。
4. 如果两个输入数组的尺寸不同，那么注意其中较大的那个尺寸。因为广播之后，两个数组的尺寸将和那个较大的尺寸一样。
5. 在任何一个维度上，如果一个数组的长度为 1，另一个数组长度大于 1，那么在该维度上，就好像是对第一个数组进行了复制。

SVM：

$$L_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + \Delta)$$

超参数在绝大多数情况下设为 1 都是安全的。超参数看起来是两个不同的超参数，但实际上他们一起控制同一个权衡：即损失函数中的数据损失和正则化损失之间的权衡。

Softmax 分类器

$$Li = -log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \text{ 或等价的 } L_i = -f_{y_i} + log(\sum_j e^{f_j})$$

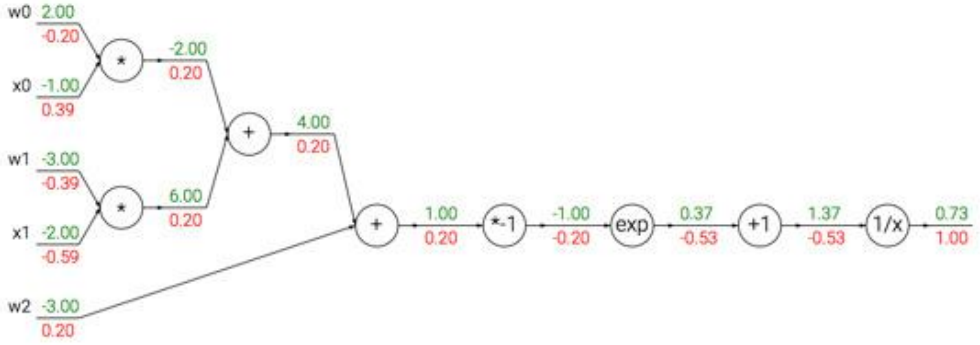
实操事项：数值稳定。编程实现 softmax 函数计算的时候，中间项因为存在指数函数，所以数值可能非常大。除以大数值可能导致数值计算的不稳定，所以学会使用归一化技巧非常重要。如果在分式的分子和分母都乘以一个常数，并把它变换到求和之中，就能

$$\frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = \frac{Ce^{f_{y_i}}}{C\sum_j e^{f_j}} = \frac{e^{f_{y_i}+logC}}{\sum_j e^{f_j+logC}}$$

得到一个从数学上等价的公式：

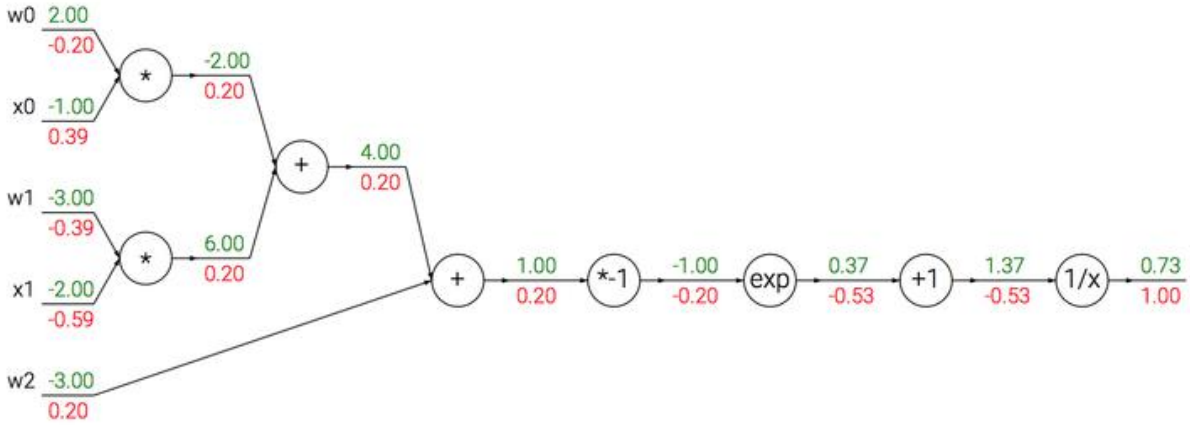
值可自由选择，不会影响计算结果，通过使用这个技巧可以提高计算中的数值稳定性。通常将最大得分值。该技巧简单地说，就是应该将向量中的数值进行平移，使得最大值为 0。

反向传播



反向传播是一个优美的局部过程。在整个计算线路图中，每个门单元都会得到一些输入并立即计算两个东西：1. 这个门的输出值，和 2. 其输出值关于输入值的局部梯度。门单元完成这两件事是完全独立的，它不需要知道计算线路中的其他细节。然而，一旦前向传播完毕，在反向传播的过程中，门单元门将最终获得整个网络的最终输出值在自己的输出值上的梯度。链式法则指出，门单元应该将回传的梯度乘以它对其的输入的局部梯度，从而得到整个网络的输出对该门单元的每个输入值的梯度。

模块化：Sigmoid 例子



模型 loss 不下降

train loss 与 test loss 结果分析

train loss 不断下降，test loss 不断下降，说明网络仍在学习;

train loss 不断下降，test loss 趋于不变，说明网络过拟合;

train loss 趋于不变，test loss 不断下降，说明数据集 100%有问题;

train loss 趋于不变，test loss 趋于不变，说明学习遇到瓶颈，需要减小学习率或批量数目;

train loss 不断上升，test loss 不断上升，说明网络结构设计不当，训练超参数设置不当，数据集经过清洗等问题。

梯度计算

计算梯度有两种方法：一个是缓慢的近似方法（**数值梯度法**），但实现相对简单。另一个方法（**分析梯度法**）计算迅速，结果精确，但是实现时容易出错，且需要使用微分。用 SVM 的损失函数在某个数据点上的计算：

$$L_i = \sum_{j \neq y_i} [max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)]$$

可以对函数进行微分。比如，对进行微分得到：

$$\nabla_{w_{y_i}} L_i = -(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0)) x_i$$

其中　　是一个示性函数，如果括号中的条件为真，那么函数值为 1，如果为假，则函数值为 0。虽然上述公式看起来复杂，但在代码实现的时候比较简单：只需要计算没有满足边界值的分类的数量（因此对损失函数产生了贡献），然后乘以就是梯度了。注意，这个梯度只是对应正确分类的 W 的行向量的梯度，那些行的梯度是：

$$\nabla_{w_j} L_i = 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

一旦将梯度的公式微分出来，代码实现公式并用于梯度更新就比较顺畅了。

梯度下降

```
# 普通的梯度下降

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # 进行梯度更新
```

小批量数据梯度下降（Mini-batch gradient descent）：

在大规模的应用中（比如 ILSVRC 挑战赛），训练数据可以达到百万级量级。如果像这样计算整个训练集，来获得仅仅一个参数的更新就太浪费了。一个常用的方法是计算训练集中的**小批量（batches）**数据

```
# 普通的小批量数据梯度下降

while True:
  data_batch = sample_training_data(data, 256) # 256个数据
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # 参数更新
```

小批量数据策略有个极端情况，那就是每个批量中只有 1 个数据样本，这种策略被称为**随机梯度下降（Stochastic Gradient Descent 简称 SGD）**，有时候也被称为在线梯度下降。这种策略在实际情况中相对少见，因为向量化操作的代码一次计算 100 个数据 比 100 次计算 1 个数据要高效很多

实验步骤：（不要求罗列完整源代码）

1. 填充相应的代码
2. 学习查找相应的 python 函数，numpy 函数
3. 测试代码运行效果
4. 思考回答相应问题

调参收敛性分析：

```
iteration 3000 / 10000: loss 2.304804
iteration 3100 / 10000: loss 2.256781
iteration 3200 / 10000: loss 1.746593
iteration 3300 / 10000: loss 1.426502
iteration 3400 / 10000: loss 1.849376
iteration 3500 / 10000: loss 1.587651
iteration 3600 / 10000: loss 1.299478
iteration 3700 / 10000: loss 1.209601
iteration 3800 / 10000: loss 1.227449
iteration 3900 / 10000: loss 1.384666
iteration 4000 / 10000: loss 1.244858
iteration 4100 / 10000: loss 1.057156
iteration 4200 / 10000: loss 1.264300
iteration 4300 / 10000: loss 0.963458
iteration 4400 / 10000: loss 1.117376
iteration 4500 / 10000: loss 0.877803
iteration 4600 / 10000: loss 1.262145
iteration 4700 / 10000: loss 1.333412
iteration 4800 / 10000: loss 1.010926
iteration 4900 / 10000: loss 0.740984
iteration 5000 / 10000: loss 0.920081
iteration 5100 / 10000: loss 0.750101
iteration 5200 / 10000: loss 1.018174
iteration 5300 / 10000: loss 0.779739
iteration 5400 / 10000: loss 0.865260
iteration 5500 / 10000: loss 0.830477
iteration 5600 / 10000: loss 0.879593
iteration 5700 / 10000: loss 0.736910
iteration 5800 / 10000: loss 0.813532
iteration 5900 / 10000: loss 0.822261
iteration 6000 / 10000: loss 0.708718
```

可以发现，需要迭代次数较多的时候才会出现收敛情况，实验时要设置的参数梯度较大，需要进行的迭代次数较多，需要使用相应的资源进行计算。

结论分析与体会：

1. 三层神经网络的梯度与反向传播

```
dscores=shift_scores_exp/shift_scores_exp_sum.reshape(-1,1)#dscores:nxc f2_out:nxh w3: hxc
dscores[range(N),y]-=1
dscores/=N
dW3=np.dot(f2_out_relu.T,dscores)+reg*W3
db3=np.sum(dscores,0)

df2_out_relu=dscores.dot(W3.T)
df2_out=(f2_out_relu>0)*df2_out_relu#nxh f1_relu_out:nxh

dW2=(f1_out_relu.T).dot(df2_out)+reg*W2
db2=np.sum(df2_out,0)

df1_out_relu=df2_out.dot(W2.T)
df1_out=(f1_out_relu>0)*df1_out_relu#nxh

dW1=(X.T).dot(df1_out)+reg*W1
db1=np.sum(df1_out,0)
```

按照维度进行相应的对应进行求解。

计算损失函数 softmax 的梯度，

2. softmax 的梯度

循　　环　　：

```
for i in range(num_train):
    scores=np.dot(X[i],W)
    scores-=np.max(scores)
    scores_exp=np.exp(scores)
    sum_score=np.sum(scores_exp)
    loss+=-scores[y[i]]+np.log(sum_score)
    for j in range(num_classes):
        if j==y[i]:
            dW[:,j]+=-X[i]
        dW[:,j]+=scores_exp[j]/sum_score*X[i]
loss/=num_train
loss+=0.5*reg*np.sum(W*W)
dW/=num_train
dW+=reg*W
```

向量化：

```
num_train=X.shape[0]
scores=np.dot(X,W)#nxc
shift_scores=scores-np.max(scores,1).reshape(-1,1)
shift_scores_exp=np.exp(shift_scores)
shift_scores_exp_sum=np.sum(shift_scores_exp,1)
loss_mat=-shift_scores[np.arange(num_train),y]+np.log(shift_scores_exp_sum)
loss=np.sum(loss_mat)
loss/=num_train
loss+=0.5*np.sum(W*W)

dS=shift_scores_exp/shift_scores_exp_sum.reshape(-1,1)
dS[np.arange(num_train),y]+=-1
dW=np.dot(X.T,dS)
dW=dW/num_train+reg*W
```

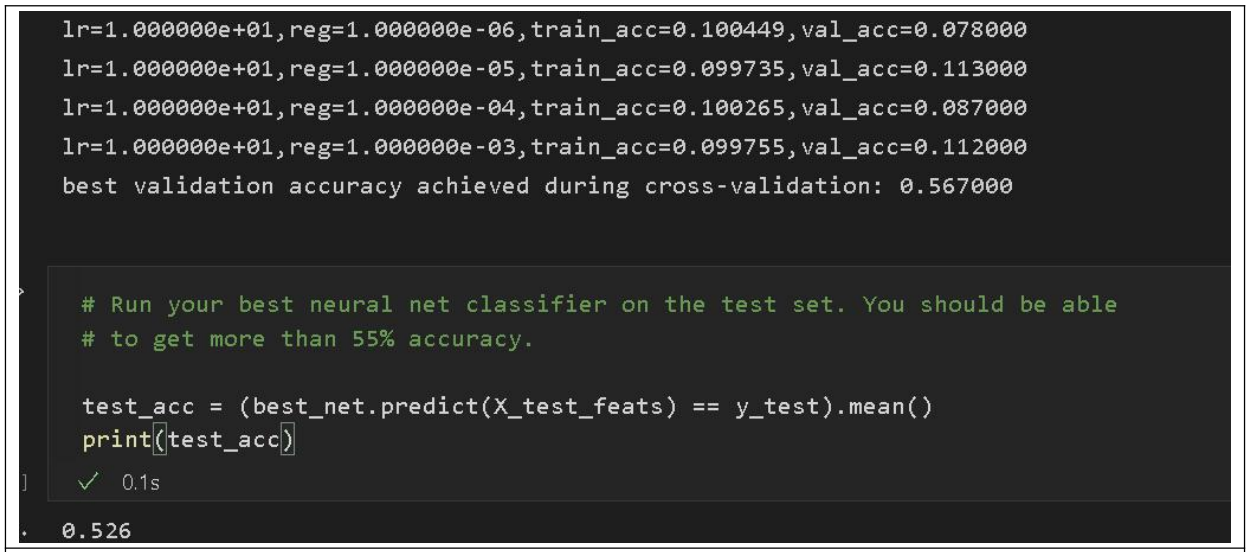
注意维度对应分析；

3. 三层神经网络的调参

```
input_size=32*32*3
hidden_size=50
num_classes=10
learning_rates=[1e-3,5e-4,5e-5,1e-4,1e-2]
regulazations=[1e-1,1e-2,1e-3,4e-3,5e-4]
# num_iters=[5000,10000]
best_val=-1
res={}
for ni in num_iters:#delete
    for lr in learning_rates:
        for reg_ in regulazations:
            net_=ThreeLayerNet(input_size,hidden_size,num_classes)
            net_.train(X_train,y_train,X_val,y_val,\
                num_iters=10000,batch_size=150,
                learning_rate=lr,learning_rate_decay=0.95,
                reg=reg_,verbose=True)
            y_train_pre=net_.predict(X_train)
            train_acc=np.mean(y_train_pre==y_train)
            y_val_pre=net_.predict(X_val)
            val_acc=np.mean(y_val_pre==y_val)
            res[(lr,reg_)]=(train_acc,val_acc)
            if val_acc>best_val:
                best_val=val_acc
                best_net=net_
for lr,rg in sorted(res):
```

不同的模型要选择不同参数，注意是独立的；

最终结果展示：



就实验过程中遇到和出现的问题，你是如何解决和处理的，自拟 1－3 道问答题：

1. 向量计算梯度复杂，手工很难推导出来：

集合维度进行分析，需要拓展的时候直接乘，需要降维的时候直接求和，其他的按照多重循环进行相应的构造。

2. 实验调参耗时长：

使用小批量测试

3. 每次调参总是出现一样的结果

每次循环内部然后建立新的模型，否则老的模型里面会有原来调参的影响，在另一个方面说明了通过增大 num_iters 可以提高模型的精度

4. python 和 numpy 使用查找官方文档

5. 计算图和梯度