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：Homework1		学号：201600122057
日期：2019.3.18	班级： 16 智能	姓名： 贝仕成
Email：337263318@qq.com		

实验目的：
完成一个对图片的 softmax 分类器和一个三层全连接层神经网络的分类器。尤其在完成后者的过程中理解反向传播和梯度下降的意义。

实验软件和硬件环境：
Python 3.5.6
Jupyter notebook 5.0.0

神舟战神 Z7M-KP7S1
NVIDIA GTX1050Ti

实验原理和方法：
前向传播 反向传播 梯度下降

实验步骤：（不要求罗列完整源代码）：
1.Softmax classifier：

这里传入的 y 都事先变成了 one-hot 向量所组成的矩阵。
Softmax_loss_vectorized（）函数的完成：

```
def softmax_loss_vectorized(W, X, y, reg):
    loss = 0.0
    dW = np.zeros_like(W)
    m=X.shape[0]
    A=np.exp(np.dot(X,W))
    A/=np.reshape(np.sum(A,axis=1),(np.array(A).shape[0],-1))
    loss=-1/m*(np.sum(np.log(A)*y))+0.5*reg*np.sum(W*W)
    dW=-1/m*np.dot(X.T,(y-A))+reg*W
    return loss, dW
```

Softmax_loss_naive()函数的完成：

```python
def softmax_loss_naive(W, X, y, reg):
    loss = 0.0
    dW = np.zeros_like(W)
    m=X.shape[0]
    A=np.dot(X,W)
    for i in range(np.array(A).shape[0]):
        A[i]=np.exp(A[i])/sum(np.exp(A[i]))
    a=np.log(A)
    J=0
    for i in range(np.array(A).shape[0]):
        J+=-(sum(a[i]*y[i]))
    J=J/m+0.5*reg*np.sum(W*W)
    dW=reg*W+(-1/m)*np.dot(X.T,(y-A))
    loss=J
    return loss, dW
```

经过测试（将 X_train 和 y_train 传入，W 随机，reg 都取 1），两函数返回的结果是一致的。

```python
import softmax as sm
sm.softmax_loss_vectorized(W, X_train, y_train, 1)
```

```
(2.6196269183007517,
 array([[  3.24401333, -11.11634402,  -1.08029171, ...,  -5.31841911,
          -7.23067865,  -6.14104472],
        [  2.35304101, -11.22662417,  -1.18732896, ...,  -5.44239348,
          -8.63679909,  -6.6248564 ],
        [  0.14330693, -11.19728046,   0.06880575, ...,  -5.11577439,
         -10.58194209,  -7.69803959],
        ...,
        [  3.81036555, -10.72796004,  -2.40819092, ...,  -6.39318763,
          -2.36491357,  -3.11245999],
        [  3.09405788, -10.59342651,  -2.5237221 , ...,  -5.80959743,
          -3.78492381,  -2.74836005],
        [  1.33612276, -10.4466402 ,  -1.60863183, ...,  -3.83232975,
          -5.36911538,  -2.97892747]]))
```

```
sm.softmax_loss_naive(W, X_train, y_train, 1)
```

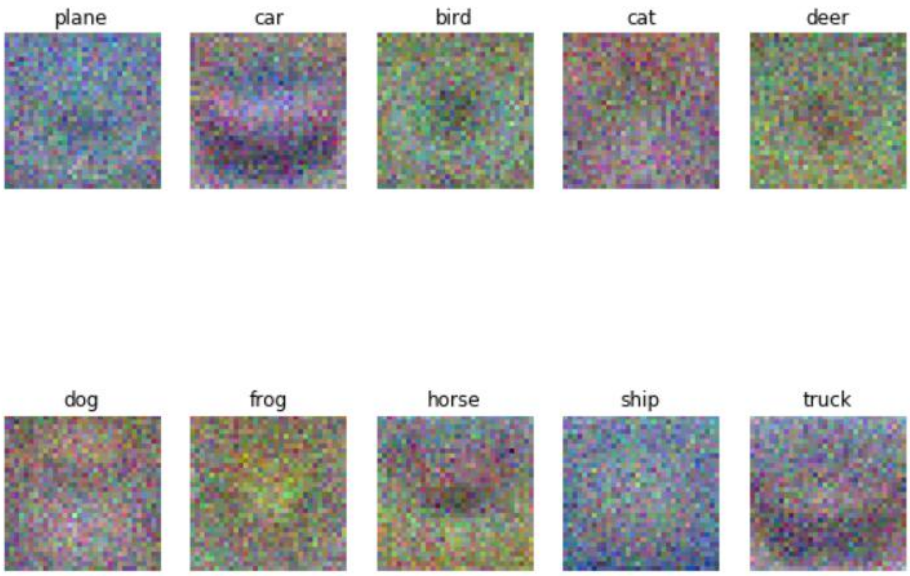
```
(2.6196269183007805,
 array([[  3.24401333, -11.11634402,  -1.08029171, ...,  -5.31841911,
          -7.23067865,  -6.14104472],
        [  2.35304101, -11.22662417,  -1.18732896, ...,  -5.44239348,
          -8.63679909,  -6.6248564 ],
        [  0.14330693, -11.19728046,   0.06880575, ...,  -5.11577439,
         -10.58194209,  -7.69803959],
        ...,
        [  3.81036555, -10.72796004,  -2.40819092, ...,  -6.39318763,
          -2.36491357,  -3.11245999],
        [  3.09405788, -10.59342651,  -2.5237221 , ...,  -5.80959743,
          -3.78492381,  -2.74836005],
        [  1.33612276, -10.4466402 ,  -1.60863183, ...,  -3.83232975,
          -5.36911538,  -2.97892747]]))
```

运行 softmax_train.py

```
Clear previously loaded data.
Train data shape:  (49000, 3073)
Train labels shape:  (49000,)
Validation data shape:  (1000, 3073)
Validation labels shape:  (1000,)
Test data shape:  (1000, 3073)
Test labels shape:  (1000,)
dev data shape:  (500, 3073)
dev labels shape:  (500,)

loss: 2.362885
sanity check: 2.302585
```

```
numerical: -1.258886 analytic: -1.258887, relative error: 1.870706e-08
numerical: 3.490080 analytic: 3.490080, relative error: 1.397187e-08
numerical: -1.325236 analytic: -1.325236, relative error: 3.311948e-09
numerical: 2.831122 analytic: 2.831122, relative error: 1.259714e-08
numerical: -2.509767 analytic: -2.509767, relative error: 5.564292e-09
numerical: 0.266541 analytic: 0.266541, relative error: 1.363033e-07
numerical: -0.383841 analytic: -0.383841, relative error: 1.169919e-07
numerical: 1.410858 analytic: 1.410858, relative error: 1.202639e-08
numerical: -1.685135 analytic: -1.685135, relative error: 1.175561e-08
numerical: 0.927501 analytic: 0.927500, relative error: 3.171021e-08
numerical: -2.500238 analytic: -2.500238, relative error: 1.335790e-08
numerical: -0.604657 analytic: -0.604657, relative error: 4.346011e-08
numerical: 3.336009 analytic: 3.336009, relative error: 2.211722e-08
numerical: 1.417428 analytic: 1.417428, relative error: 2.718325e-08
numerical: -0.139882 analytic: -0.139882, relative error: 3.960532e-07
numerical: 6.519727 analytic: 6.519727, relative error: 1.857496e-08
numerical: -0.024219 analytic: -0.024219, relative error: 1.734651e-06
numerical: 1.282356 analytic: 1.282356, relative error: 2.747164e-08
numerical: 0.393403 analytic: 0.393403, relative error: 1.728310e-08
numerical: -0.808343 analytic: -0.808343, relative error: 1.713314e-08
naive loss: 2.362885e+00 computed in 0.014962s
vectorized loss: 2.362885e+00 computed in 0.014964s
Loss difference: 0.000000
Gradient difference: 0.000000
```

```
lr 1.000000e-07 reg 2.500000e+04 train accuracy: 0.348776 val accuracy: 0.357000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.323367 val accuracy: 0.339000
lr 5.000000e-07 reg 2.500000e+04 train accuracy: 0.343490 val accuracy: 0.355000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.321714 val accuracy: 0.337000
best validation accuracy achieved during cross-validation: 0.357000
softmax on raw pixels final test set accuracy: 0.353000
```



2. Three_layer_net classifier:

neural_net.py:

Forward pass:

```
T=np.dot(X,W1)+b1
T1=np.maximum(T,0)
T=np.dot(T1,W2)+b2
T2=np.maximum(T,0)
scores=np.dot(T2,W3)+b3
```

计算 loss:

```
exp_scores=np.exp(scores)
exp_scores/=(np.sum(exp_scores,axis=1).reshape(N,1))
loss=-(1/N)*(np.sum(np.log(exp_scores[np.arange(N),y])))+0.5*reg*np.sum(W1*W1)+0.5*reg*np.sum(W2*W2)+0.5*reg*np.sum(W3*W3)
```

Backward propagation:(这里用 computational graph 可以更好地理解 BP 的计算过程,计算对各个矩阵（数组）的梯度是为了方便后面使用梯度下降来进行相应的更新）

```
delta_S=np.zeros_like(exp_scores)
delta_S[range(N),y]+=1
delta_S-=exp_scores
grads = {}

grads['W3']=reg*W3+(-1/N)*np.dot(T2.T,delta_S)
grads['b3']=-(1/N)*np.sum(delta_S,axis=0)


delta_t2=np.zeros_like(T2)
delta_t2[T2>0]=1


grads['W2']=reg*W2+(-1/N)*np.dot(T1.T,np.dot(delta_S,W3.T)*delta_t2)
grads['b2']=(-1/N)*np.sum(np.dot(delta_S,W3.T)*delta_t2,axis=0)

delta_t1 = np.zeros_like(T1)

zhenghe=(np.dot(delta_S,W3.T))*delta_t2
delta_t1[T1>0]=1

grads['W1']=reg*W1+(-1/N)*np.dot(X.T,np.dot(zhenghe,W2.T)*delta_t1)
grads['b1']=(-1/N)*np.sum(np.dot(zhenghe,W2.T)*delta_t1,axis=0)
```

创建 minibatch:

```
for it in range(num_iters):
    r=np.random.choice(num_train,batch_size)
    X_batch=X[r,:]
    y_batch=y[r]
```

靠梯度下降来更新参数：

```
loss, grads = self.loss(X_batch, y=y_batch, reg=reg)
loss_history.append(loss)

self.params['W1']-=learning_rate*grads['W1']
self.params['W2']-=learning_rate*grads['W2']
self.params['W3']-=learning_rate*grads['W3']
self.params['b1']-=learning_rate*grads['b1']
self.params['b2']-=learning_rate*grads['b2']
self.params['b3']-=learning_rate*grads['b3']
```

预测函数 predict()的完成：
在每个样本经过三层全连接层后经过 softmax 层所得出的十维结果中，找到值最大的那一维分量所对应的位置即为其所被预测的标签。

```
def predict(self, X):
        y_pred = None
        score=self.loss(X)
        y_pred=np.argmax(score,axis=1)
        return y_pred
```

three_layer_net.py:
超参的设置：（这里先经过了一次预训练，筛选出了可能合适的参数选项，由于跑的较慢，后选择挂在服务器上跑）

```
hids=[256,512,1024]
lrs=[1e-2,5e-3,1e-3]
rgs=[1e-2,1e-3,1e-4]
batch_sizes=[100,200]
```

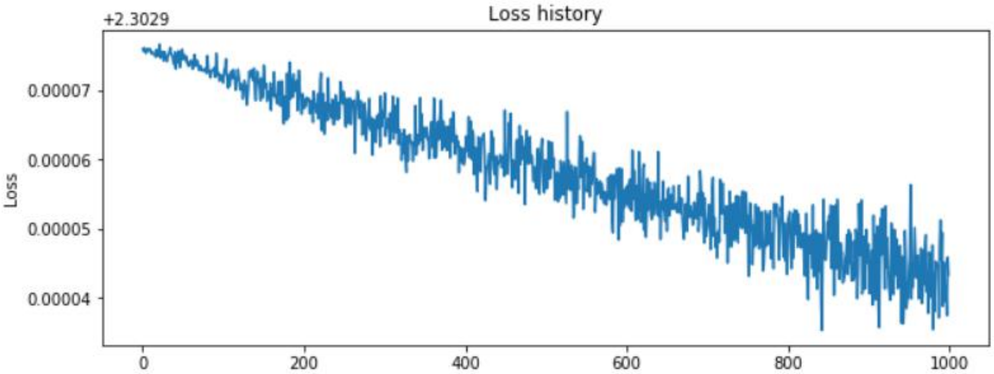
不断地训练，最终返回一个最好的网络及其对验证集的准确率：

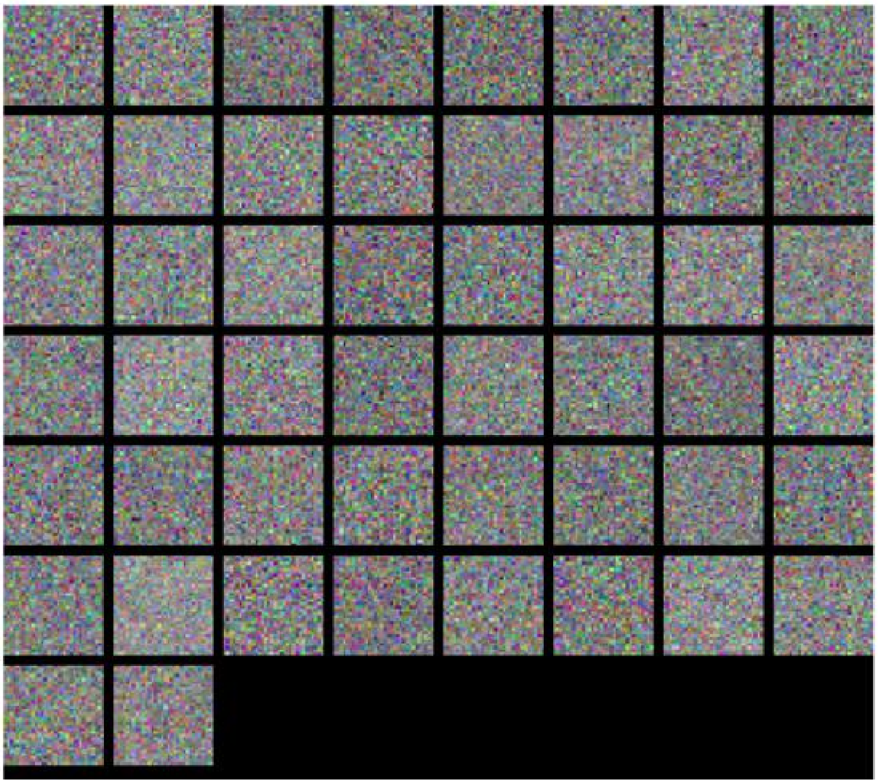
```
for hid in hids:
    for lr in lrs:
        for rg in rgs:
            for bt in batch_sizes:
                print("hid:%d,lr:%.4f,reg:%.4f,batch_size:%d" %(hid,lr,rg,bt))
                net=ThreeLayerNet(input_size,hid,num_classes)
                status=net.train(X_train,y_train,X_val,y_val,num_iters=5000,batch_size=bt,
                                 learning_rate=lr,learning_rate_decay=0.98,
                                 reg=rg,verbose=True)
                acc_train_val=status['train_acc_history'][-1]
                acc_validation_val=status['val_acc_history'][-1]
                print("train accuracy:%.4f" %acc_train_val)
                print("validation accuracy:%.4f" %acc_validation_val)

                if acc_validation_val>best_val:
                    best_val=acc_validation_val
                    best_net=net
                    best_status=status
                    best_para_detail=(hid,lr,rg,bt)

print("best validation accuracy:%.4f" %best_val)
print(best_para_detail)
print(net.get_param())
```

```
show_net_weights(best_net)
```

```
test_acc = (best_net.predict(X_test) == y_test).mean()
print('Test accuracy: ', test_acc)
```

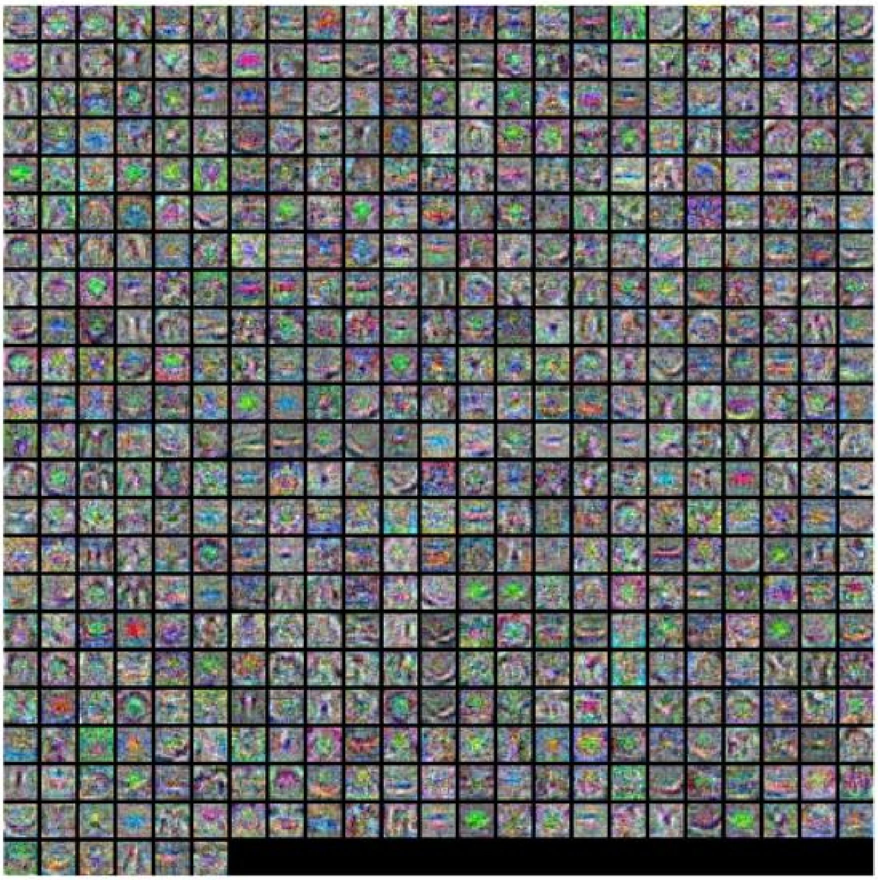
实验结果：





训练出的最好网络（相应参数为 **hidden_size,learning_rate,reg,batch_size**）：

```
best validation accuracy:0.5730
(512, 0.01, 0.001, 200)
```



Weights 可视化

测试集准确率：

Test accuracy: 0.547

已经达到实验的基本要求。

结论分析与体会：

1. 实现代码所需的基本数学知识很重要，对矩阵求导都不熟的话，很容易耽搁时间，之后代码还不一定敲得对；
2. 参数的设置对实验结果的影响很大，就拿和 minibatch 有关的 num_iters 来说，num_iters 是每一种网络训练的时候 batch 被送进训练的次数，因为 batch 是随机的，所以存在样本被重复送入训练的情况，这一次训练中可能整个训练集已经被送进去了好几次。这个如果不够高的话那么训练出的网络一定不会优秀，num_iters 我设的 5000，明显就比设 1000 时结果好得多。

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. 如何正确计算反向传播过程中对最终输出结果 C 对各个 W、b 的梯度？

这里强烈推荐画 computational graph 来进行理解，可以清晰地看到每一个 W、b 到 C 的路径，从 C 出发，每一个节点对后一个节点的梯度都可以很快计算出来，因此把选择的路径上每一段的矩阵连乘就可以得到对 W、b 的梯度了。这个过程一定要注意连乘要严格遵循先后顺序，且一些矩阵要经过转置才能得到正确结果。

2. 关于调参有什么技巧？

参数并不是每次都要自己手动调一次训练一次，那样很麻烦。直接列下各个超参的可能取值，靠 for 循环来将每个可能的超参组合遍历一遍来训练网络，通过不断的对比更新最优的网络参数，最后只讲这最优的网络保留，并得到它对验证集，测试集的准确率。注意各个参数取值的 list 不应设得过长，合理挑范围设置即可，不然可是要跑很久的。除非自己的算力很庞大，不然建议先粗略的选取范围，比较结果后在逐渐缩小可调参的范围。