

最近在刷cs231n的课程和作业，在这里分享下自己的学习过程，同时也希望能够得到大家的指点。

## 写在前面：

1. 这仅仅是自己的学习笔记，如果侵权，还请告知；
2. 代码是参照[lightaime的github](#)，在其基础之上做了一些修改；
3. 在我之前，已有前辈在他的[知乎](#)上分享过类似内容；
4. 讲义是参照[杜客](#)等人对cs231n的中文翻译。

## 温馨提醒：

1. 这篇文档是建立在你已经知道神经网络的基本原理，如果在阅读本篇文档过程中有些原理不是很清楚，请移步温习下相关知识（参阅cs231n讲义）；
2. 文档的所有程序是使用python3.5实现的，如果你是python2的用户，可能要对代码稍作修改；
3. 文章频繁提到将代码保存到 `.py` 中，是为了方便接下来的模块导入，希望读者可以理解。

使用神经网络完成cifa10的分类工作，主要有以下几个内容：

1. 神经网络模型构建；
2. 数据处理；
3. 训练；
4. 预测。

# 1 两层神经网络模型的构建

## 1.1 损失函数和梯度

我们先来看下每一层神经网络的表达式

第一层神经网络表达式，这里激活函数采用的是Relu：

$$z_1 = w_1 x + b_1$$

$$h = \max(0, z_1) \quad \text{Relu}$$

第二层神经网络表达式：

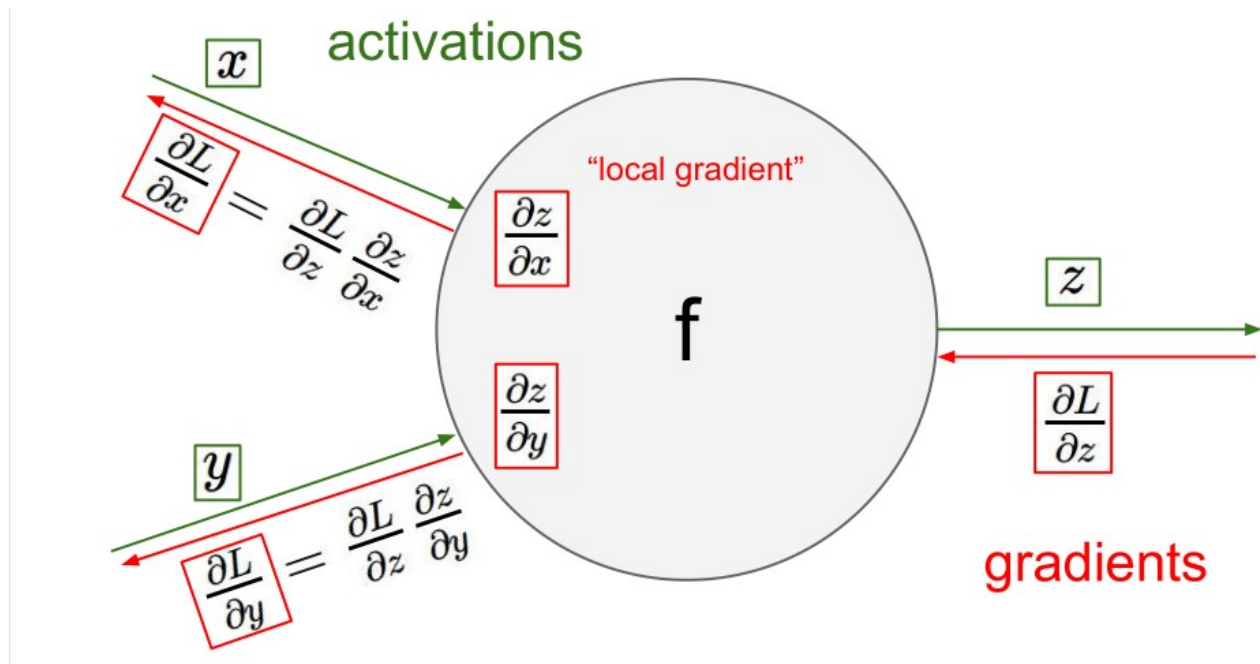
$$z_2 = w_2 h + b_2$$

$$f(z_2) = \frac{e^z}{\sum e^z} \quad \text{Softmax}$$

其中第二层采用softmax作为激活函数。

根据上式我们可以计算出每一层的输出。

梯度计算是利用方向传播算法计算的，也就是说，第一层的梯度是在第二层梯度的基础之上计算出来的，这里我们可以放一张cs231n中的ppt。



绿色表示的loss值，红色表示的是梯度值。

代码如下：`neural_net.py`

```
import numpy as np
class TwoLayerNet(object):
```

```
1 def __init__(self, input_size, hidden_size, output_size, std=1e-4):
2     self.params={}
3     self.params['W1']=std*np.random.randn(input_size,hidden_size)
4     self.params['b1']=np.zeros(hidden_size)
5     self.params['W2']=std*np.random.randn(hidden_size,output_size)
6     self.params['b2']=np.zeros(output_size)
7
8
9 def loss(self,X,y=None,reg=0.0):
10
11     W1,b1=self.params['W1'],self.params['b1']
12     W2,b2=self.params['W2'],self.params['b2']
13     N,D=X.shape
14
15     scores=None
16
```

```

17     h_output=np.maximum(0,X.dot(W1)+b1) #第一层输出(N,H)，Relu激活函数
18     scores=h_output.dot(W2)+b2 #第二层激活函数前的输出(N,C)
19
20     if y is None:
21         return scores
22     loss=None
23
24     shift_scores=scores-np.max(scores,axis=1).reshape((-1,1))
25
26     softmax_output=np.exp(shift_scores)/np.sum(np.exp(shift_scores),axis=1).
27     reshape(-1,1)
28     loss=-np.sum(np.log(softmax_output[range(N),list(y)]))
29     loss/=N
30     loss+=0.5*reg*(np.sum(W1*W1)+np.sum(W2*W2))#正则项
31
32     grads={}
33
34     #第二层梯度计算
35     dscores=softmax_output.copy()
36     dscores[range(N),list(y)]=-1
37     dscores/=N
38     grads['W2']=h_output.T.dot(dscores)+reg*W2
39     grads['b2']=np.sum(dscores,axis=0)
40
41     #第一层梯度计算
42     dh = dscores.dot(W2.T)
43     dh_ReLu = (h_output > 0) * dh
44     grads['W1'] = X.T.dot(dh_ReLu) + reg * W1
45     grads['b1'] = np.sum(dh_ReLu, axis=0)
46
47     return loss,grads

```

这里，我们说下对输入数据的要求  $X$  (N,D)维输入数据,  $y$  (N,)标签,  $W1$  (D,H)第一层权重,  $b1$  (H,)第一层偏置,  $W2$  (H,C)第二层权重,  $b2$  (C,)第二层偏置

上面的代码重点是反向传播阶段，输入是图像数据时，我们都是对矩阵和向量进行操作，然而，在操作的时候，要注意关注维度和转置操作。这里我们引用下cs231n中讲义的解释：

**提示：要分析维度！** 注意不需要去记忆 $dW$ 和 $dX$ 的表达，因为它们很容易通过维度推导出来。例如，权重的梯度 $dW$ 的尺寸肯定和权重矩阵 $W$ 的尺寸是一样的，而这又是由 $X$ 和 $dD$ 的矩阵乘法决定的（在上面的例子中 $X$ 和 $W$ 都是数字不是矩阵）。总有一个方式是能够让维度之间能够对的上的。例如， $X$ 的尺寸是 $[10 \times 3]$ ， $dD$ 的尺寸是 $[5 \times 3]$ ，如果你想要 $dW$ 和 $W$ 的尺寸是 $[5 \times 10]$ ，那就要 $dD.dot(X.T)$ 。

## 1.2 训练和预测模型

接下来是训练和预测模型。

我们依旧使用SGD进行训练，并将训练过程中的每一轮的平均损失函数、平均训练准确率、平均验证准确率保存

在 `loss_history` , `train_acc_history` , `val_acc_history` 中。实现的代码如下：

```
1 def train(self,X,y,X_val,y_val,learning_rate=1e-
2     3,learning_rate_decay=0.95,reg=1e-5,num_iters=100,
3         batch_size=200,verbose=False):
4     num_train=X.shape[0]
5     iterations_per_epoch=max(num_train/ batch_size,1) #每一轮迭代数目
6     loss_history=[]
7     train_acc_history=[]
8     val_acc_history=[]
9
10    for it in range(num_iters):
11        X_batch=None
12        y_batch=None
13
14        idx=np.random.choice(num_train,batch_size,replace=True)
15        X_batch=X[idx]
16        y_batch=y[idx]
17        loss,grads=self.loss(X_batch,y=y_batch,reg=reg)
18        loss_history.append(loss)
19
20        #参数更新
21        self.params['W2']+=-learning_rate*grads['W2']
22        self.params['b2']+=-learning_rate*grads['b2']
23        self.params['W1']+=-learning_rate*grads['W1']
24        self.params['b1']+=-learning_rate*grads['b1']
25
26        if verbose and it % 100 ==0: #每迭代100次，打印
27            print('iteration %d / %d : loss %f' % (it,num_iters,loss))
28
29        if it % iterations_per_epoch==0: #一轮迭代结束
30            train_acc=(self.predict(X_batch)==y_batch).mean()
31            val_acc=(self.predict(X_val)==y_val).mean()
32            train_acc_history.append(train_acc)
33            val_acc_history.append(val_acc)
34
35        #更新学习率
36        learning_rate*=learning_rate_decay
```

```

36     return {
37         'loss_history':loss_history,
38         'train_acc_history':train_acc_history,
39         'val_acc_history':val_acc_history
40     }

```

之后，我们只需要使用训练保存好的参数，就可以完成预测，代码如下：

```

1 def predict(self,X):
2     y_pred=None
3     h=np.maximum(0,X.dot(self.params['W1'])+self.params['b1'])
4     scores=h.dot(self.params['W2'])+self.params['b2']
5     y_pred=np.argmax(scores,axis=1)
6     return y_pred

```

以上所有代码我们保存在 `neural_net.py` 文件中。

这样，我们就完成了两层神经网络模型的构建工作。

## 2 数据处理 `DataPreprocess.py`

数据处理部分，同前面我们所叙述的，先将cifar10图片转成数组，然后对其进行归一化处理（减去均值）。因为前面已经叙述很多，我们这里就不再赘述，只是将代码给大家贴出来：

```

import numpy as np
from data_utils import load_cifar10

1 def get_cifar_data(num_training=49000,num_validation=1000,num_test=1000):
2     cifar10_dir='../knn/cifar-10-batches-py/datasets'
3     X_train,y_train,X_test,y_test=load_cifar10(cifar10_dir)
4     # 验证集
5     mask=range(num_training,num_training+num_validation)
6     X_val=X_train[mask]
7     y_val=y_train[mask]
8     #训练集
9     mask=range(num_training)
10    X_train=X_train[mask]
11    y_train=y_train[mask]
12    #测试集
13    mask=range(num_test)
14    X_test=X_test[mask]
15    y_test=y_test[mask]
16
17    mean_image=np.mean(X_train,axis=0)
18    X_train-=mean_image

```

```

19     X_val-=mean_image
20     X_test-=mean_image
21
22     X_train=X_train.reshape(num_training,-1)
23     X_val=X_val.reshape(num_validation,-1)
24     X_test=X_test.reshape(num_test,-1)
25
26     return X_train,y_train,X_val,y_val,X_test,y_test

```

得到结果如下：

```

1 train data shape: (49000, 3072)
2 train labels shape: (49000,)
3 validation data shape: (1000, 3072)
4 validation labels shape: (1000,)
5 test data shape: (1000, 3072)
6 test labels shape: (1000,)

```

# 验证结果是否正确

```

X_train,y_train,X_val,y_val,X_test,y_test = get_cifar_data()
print("\n验证分离验证集结果是否正确")
print('training data shape:', X_train.shape)
print('training labels shape:', y_train.shape)
print('validation data shape:', X_val.shape)
print('validation data shape:', y_val.shape)
print('test data shape:', X_test.shape)
print('test labels shape:', y_test.shape)

```

## 3 模型训练

### 3.1 网络初始训练 `train.py`

我们使用带有动量的SGD来训练我们的网络，也就是说，我们在每一轮结束之后更新我们的学习速率，这里我们是乘上一个衰减率。

代码如下：

```

from neural_net import TwoLayerNet
from DataPreprocess import *
1 input_size=32*32*3
2 hidden_size=50
3 num_classes=10
4 net=TwoLayerNet(input_size,hidden_size,num_classes)
5 stats=net.train(X_train,y_train,X_val,y_val,num_iters=1000,batch_size=200
,learning_rate=1e-4,learning_rate_decay=0.95,
6                 reg=0.5,verbose=True)
7 val_acc=(net.predict(X_val)==y_val).mean()
8 print('validation accuracy:',val_acc)

```

输出结果如下：

```

1 iteration 0 / 1000 : loss 2.302975
2 iteration 100 / 1000 : loss 2.302409

```

```
3 iteration 200 / 1000 : loss 2.297453
4 iteration 300 / 1000 : loss 2.274700
5 iteration 400 / 1000 : loss 2.211710
6 iteration 500 / 1000 : loss 2.126385
7 iteration 600 / 1000 : loss 2.074668
8 iteration 700 / 1000 : loss 2.056960
9 iteration 800 / 1000 : loss 2.002378
10 iteration 900 / 1000 : loss 2.004737
11 valiadation accuracy: 0.279
```

0.28的准确率其实并不是很好，我们需要知道中间发生了什么，让我们的结果变得不理想，这里有两种方法：

1. 可视化loss和accuracy；
2. 可视化权重。

我们来分别实现下。

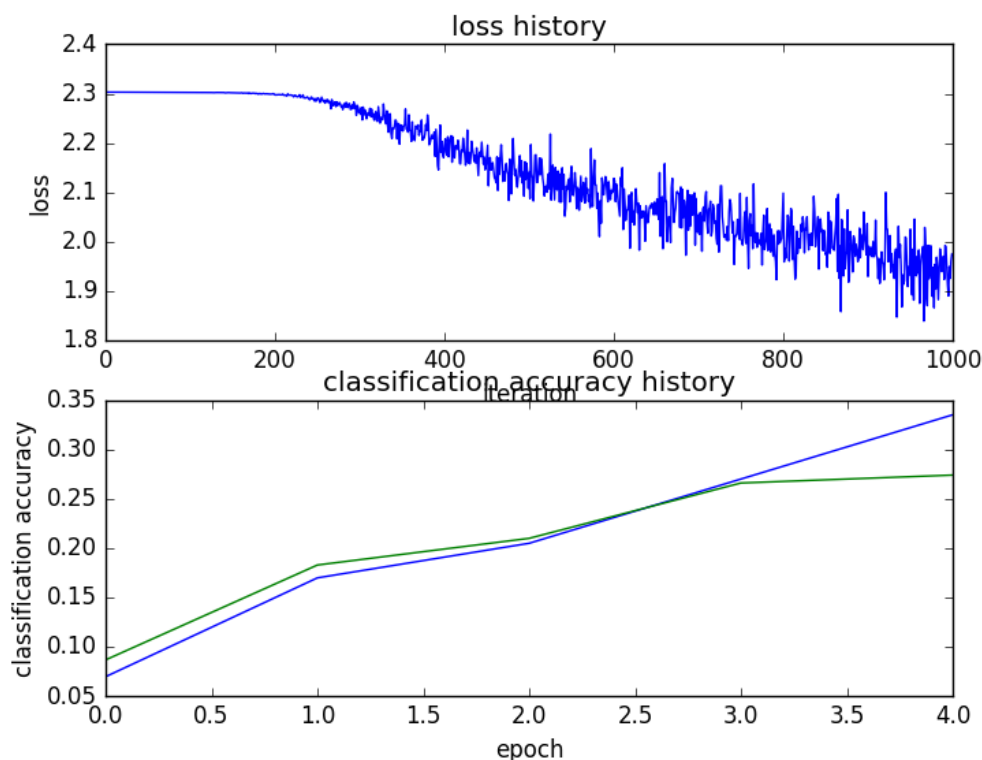
### 3.1.1 loss和accura可视化 `vis_utils.py`

代码如下：

```
import matplotlib.pyplot as plt
from train import stats

1 plt.subplot(211)
2 plt.plot(stats['loss_history'])
3 plt.title('loss history')
4 plt.xlabel('iteration')
5 plt.ylabel('loss')
6
7 plt.subplot(212)
8 plt.plot(stats['train_acc_history'],label='train')
9 plt.plot(stats['val_acc_history'],label='val')
10 plt.title('classification accuracy history')
11 plt.xlabel('epoch')
12 plt.ylabel('classification accuracy')
13 plt.show()
```

结果如下：



### 3.1.2 权重可视化

cs231n为我们提供好了可以权重可视化的函数，我们直接调用就好，不过我们还是看下看下这份代码：

```

1  from math import ceil, sqrt
   import numpy as np
2  def visualize_grid(Xs,ubound=255.0,padding=1):
3      (N,H,W,C)=Xs.shape
4      grid_size=int(ceil(sqrt(N)))
5      grid_height=H*grid_size+padding*(grid_size-1)
6      grid_width=W*grid_size+padding*(grid_size-1)
7      grid=np.zeros((grid_height,grid_width,C))
8      next_idx=0
9      y0,y1=0,H
10     for y in range(grid_size):
11         x0,x1=0,W
12         for x in range(grid_size):
13             if next_idx<N:
14                 img=Xs[next_idx]
15                 low,high=np.min(img),np.max(img)
16                 grid[y0:y1,x0:x1]=ubound*(img-low)/(high-low)
17                 next_idx+=1
18             x0+=W+padding
19             x1+=W+padding
20         y0+=H+padding
21         y1+=H+padding

```



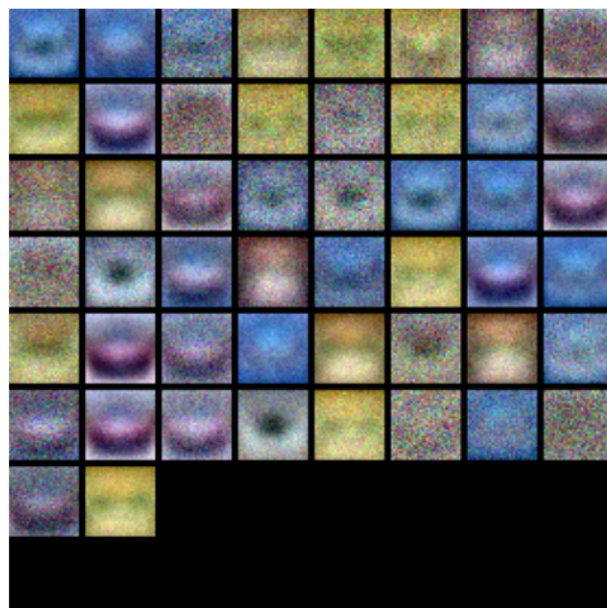
这份代码要求我们输入的 `Xs` 是四维的。

我将这份代码保存在了 `vis_utils.py` 中。

我们调用它，来可视化下权重：

```
from train import net
1 def show_net_weights(net):
2     W1=net.params['W1']
3     W1=W1.reshape(32,32,3,-1).transpose(3,0,1,2)
4     plt.imshow(visualize_grid(W1,padding=3).astype('uint8'))
5     plt.axis('off')
6     plt.show()
7
8 show_net_weights(net)
```

可视化结果如下：



## 3.2 超参数调整

在调整超参数之前，我们先来分析下我们上面得到的可视化结果。

我们的loss曲线大致呈线性下降，这表明我们设置的学习率可能太低；训练集和验证集的准确率很接近，表明我们的模型复杂度不够，也就是欠拟合。（当然增加模型复杂度的话可能会导致过拟合。）

下面我们就通过交叉验证的方法来不断调整超参数。

代码如下：

```
import numpy as np
from train import *
```

```
1 input_size=input_size=32*32*3
2 num_classes=10
3 hidden_size=[75,100,125]
4 results={}
5 best_val_acc=0
6 best_net=None
7
8 learning_rates=np.array([0.7,0.8,0.9,1.0,1.1])*1e-3
9 regularization_strengths=[0.75,1.0,1.25]
10 print('running')
11 for hs in hidden_size:
12     for lr in learning_rates:
13         for reg in regularization_strengths:
14             net=TwoLayerNet(input_size,hs,num_classes)
15
16             stats=net.train(X_train,y_train,X_val,y_val,num_iters=1500,batch_size=20
0,
17                             learning_rate=lr,learning_rate_decay=0.95,
18                             reg=reg,verbose=False)
19             val_acc=(net.predict(X_val)==y_val).mean()
20             if val_acc >best_val_acc:
21                 best_val_acc=val_acc
22                 best_net=net
23                 results[(hs,lr,reg)]=val_acc
24
25 print('finshed')
26
27 for hs,lr,reg in sorted(results):
28     val_acc=results[(hs,lr,reg)]
29     print('hs %d lr %e reg %e val accuracy: %f' % (hs,lr,reg,val_acc))
30
31 print('best validation accuracy achieved during cross_validation: %f' %
    best_val_acc)
```

输出结果如下：

```
1 hs 75 lr 7.000000e-04 reg 7.500000e-01 val accuracy: 0.489000
2 hs 75 lr 7.000000e-04 reg 1.000000e+00 val accuracy: 0.475000
3 hs 75 lr 7.000000e-04 reg 1.250000e+00 val accuracy: 0.479000
4 hs 75 lr 8.000000e-04 reg 7.500000e-01 val accuracy: 0.475000
5 hs 75 lr 8.000000e-04 reg 1.000000e+00 val accuracy: 0.490000
6 hs 75 lr 8.000000e-04 reg 1.250000e+00 val accuracy: 0.485000
7 hs 75 lr 9.000000e-04 reg 7.500000e-01 val accuracy: 0.496000
```

```
8 hs 75 lr 9.000000e-04 reg 1.000000e+00 val accuracy: 0.494000
9 hs 75 lr 9.000000e-04 reg 1.250000e+00 val accuracy: 0.487000
10 hs 75 lr 1.000000e-03 reg 7.500000e-01 val accuracy: 0.471000
11 hs 75 lr 1.000000e-03 reg 1.000000e+00 val accuracy: 0.495000
12 hs 75 lr 1.000000e-03 reg 1.250000e+00 val accuracy: 0.495000
13 hs 75 lr 1.100000e-03 reg 7.500000e-01 val accuracy: 0.483000
14 hs 75 lr 1.100000e-03 reg 1.000000e+00 val accuracy: 0.492000
15 hs 75 lr 1.100000e-03 reg 1.250000e+00 val accuracy: 0.475000
16 hs 100 lr 7.000000e-04 reg 7.500000e-01 val accuracy: 0.476000
17 hs 100 lr 7.000000e-04 reg 1.000000e+00 val accuracy: 0.483000
18 hs 100 lr 7.000000e-04 reg 1.250000e+00 val accuracy: 0.477000
19 hs 100 lr 8.000000e-04 reg 7.500000e-01 val accuracy: 0.477000
20 hs 100 lr 8.000000e-04 reg 1.000000e+00 val accuracy: 0.480000
21 hs 100 lr 8.000000e-04 reg 1.250000e+00 val accuracy: 0.493000
22 hs 100 lr 9.000000e-04 reg 7.500000e-01 val accuracy: 0.504000
23 hs 100 lr 9.000000e-04 reg 1.000000e+00 val accuracy: 0.492000
24 hs 100 lr 9.000000e-04 reg 1.250000e+00 val accuracy: 0.472000
25 hs 100 lr 1.000000e-03 reg 7.500000e-01 val accuracy: 0.511000
26 hs 100 lr 1.000000e-03 reg 1.000000e+00 val accuracy: 0.490000
27 hs 100 lr 1.000000e-03 reg 1.250000e+00 val accuracy: 0.488000
28 hs 100 lr 1.100000e-03 reg 7.500000e-01 val accuracy: 0.510000
29 hs 100 lr 1.100000e-03 reg 1.000000e+00 val accuracy: 0.499000
30 hs 100 lr 1.100000e-03 reg 1.250000e+00 val accuracy: 0.480000
31 hs 125 lr 7.000000e-04 reg 7.500000e-01 val accuracy: 0.496000
32 hs 125 lr 7.000000e-04 reg 1.000000e+00 val accuracy: 0.482000
33 hs 125 lr 7.000000e-04 reg 1.250000e+00 val accuracy: 0.466000
34 hs 125 lr 8.000000e-04 reg 7.500000e-01 val accuracy: 0.460000
35 hs 125 lr 8.000000e-04 reg 1.000000e+00 val accuracy: 0.466000
36 hs 125 lr 8.000000e-04 reg 1.250000e+00 val accuracy: 0.479000
37 hs 125 lr 9.000000e-04 reg 7.500000e-01 val accuracy: 0.502000
38 hs 125 lr 9.000000e-04 reg 1.000000e+00 val accuracy: 0.499000
39 hs 125 lr 9.000000e-04 reg 1.250000e+00 val accuracy: 0.468000
40 hs 125 lr 1.000000e-03 reg 7.500000e-01 val accuracy: 0.489000
41 hs 125 lr 1.000000e-03 reg 1.000000e+00 val accuracy: 0.475000
42 hs 125 lr 1.000000e-03 reg 1.250000e+00 val accuracy: 0.509000
43 hs 125 lr 1.100000e-03 reg 7.500000e-01 val accuracy: 0.481000
44 hs 125 lr 1.100000e-03 reg 1.000000e+00 val accuracy: 0.477000
45 hs 125 lr 1.100000e-03 reg 1.250000e+00 val accuracy: 0.480000
46 best validation accuracy achieved during cross_validation: 0.511000
```

可以看到，在验证集上最好的识别准确率达到了0.511，这比我们前面说的几个分类器高出很多了，而且我们才仅仅使用了一个隐层，可见神经网络功能的强大。我们将验证集准确率最高的模型保存在了 `best_net`，以便我们接下来可以直接使用该模型进行预测。

我们也看下我们的神经网络学习到了什么，如下：



同样的第一层的权重，这次得到的特征就比上次我们得到的特征清晰很多了。

## 4 预测

我们直接可以使用上面保存好的 `best_net` 进行预测，代码如下：

```
from best_net import *  
1 test_acc=(best_net.predict(X_test)==y_test).mean()  
2 print('test accuracy:' , test_acc)
```

最后得到的平均准确率如下：

```
1 test accuracy: 0.502
```

## 5 总结

这一节我们构建了一个两层的神经（全连接）网络，并使用该网络完成了对cifar10数据的分类工作。

神经网络和我们前面介绍的分类器不同的是，它的梯度更新方式是利用反向传播算法，反向传播算法是根据链式求导法则来计算的。