Softmax 练习

补充并完成本练习。

本练习类似于SVM练习，你要完成的事情包括:

- 为Softmax分类器实现完全矢量化的**损失函数**
- 实现其**解析梯度（analytic gradient）**的完全矢量化表达式
- 用数值梯度**检查你的代码**
- 使用验证集**调整学习率和正则化强度**
- 使用**SGD优化**损失函数
- **可视化**最终学习的权重

In [1]:

```
import random
import numpy as np
from daseCV.data_utils import load_CIFAR10
import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading extenrnal modules
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

In [2]:

```
def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000, num_dev=
    """
    Load the CIFAR-10 dataset from disk and perform preprocessing to prepare
    it for the linear classifier. These are the same steps as we used for the
    SVM, but condensed to a single function.
    """
    # Load the raw CIFAR-10 data
    cifar10_dir = 'daseCV/datasets/cifar-10-batches-py'

    # Cleaning up variables to prevent loading data multiple times (which may cause me
    try:
       del X_train, y_train
       del X_test, y_test
       print('Clear previously loaded data.')
    except:
       pass

    X_train, y_train, X_test, y_test = load_CIFAR10(cifar10_dir)

    # subsample the data
    mask = list(range(num_training, num_training + num_validation))
    X_val = X_train[mask]
    y_val = y_train[mask]
    mask = list(range(num_training))
    X_train = X_train[mask]
    y_train = y_train[mask]
    mask = list(range(num_test))
    X_test = X_test[mask]
```

```
    y_test = y_test[mask]
    mask = np.random.choice(num_training, num_dev, replace=False)
    X_dev = X_train[mask]
    y_dev = y_train[mask]

    # Preprocessing: reshape the image data into rows
    X_train = np.reshape(X_train, (X_train.shape[0], -1))
    X_val = np.reshape(X_val, (X_val.shape[0], -1))
    X_test = np.reshape(X_test, (X_test.shape[0], -1))
    X_dev = np.reshape(X_dev, (X_dev.shape[0], -1))

    # Normalize the data: subtract the mean image
    mean_image = np.mean(X_train, axis = 0)
    X_train -= mean_image
    X_val -= mean_image
    X_test -= mean_image
    X_dev -= mean_image

    # add bias dimension and transform into columns
    X_train = np.hstack([X_train, np.ones((X_train.shape[0], 1))])
    X_val = np.hstack([X_val, np.ones((X_val.shape[0], 1))])
    X_test = np.hstack([X_test, np.ones((X_test.shape[0], 1))])
    X_dev = np.hstack([X_dev, np.ones((X_dev.shape[0], 1))])

    return X_train, y_train, X_val, y_val, X_test, y_test, X_dev, y_dev


# Invoke the above function to get our data.
X_train, y_train, X_val, y_val, X_test, y_test, X_dev, y_dev = get_CIFAR10_data()
print('Train data shape: ', X_train.shape)
print('Train labels shape: ', y_train.shape)
print('Validation data shape: ', X_val.shape)
print('Validation labels shape: ', y_val.shape)
print('Test data shape: ', X_test.shape)
print('Test labels shape: ', y_test.shape)
print('dev data shape: ', X_dev.shape)
print('dev labels shape: ', y_dev.shape)
```

```
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,)
```

Softmax 分类器

请在**daseCV/classifiers/softmax.py**中完成本节的代码。

In [3]:

```
# 首先使用嵌套循环实现简单的softmax损失函数。
# 打开文件 daseCV/classifiers/softmax.py 并补充完成
# softmax_loss_naive 函数.

from daseCV.classifiers.softmax import softmax_loss_naive
import time

# 生成一个随机的softmax权重矩阵，并使用它来计算损失。
W = np.random.randn(3073, 10) * 0.0001
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# As a rough sanity check, our loss should be something close to -log(0.1).
```

```
print('loss: %f' % loss)
print('sanity check: %f' % (-np.log(0.1)))
```

```
loss: 2.340605
sanity check: 2.302585
```

问题 1

为什么我们期望损失接近-log（0.1）？简要说明。

答：*在这里写上你的答案*

由于权重矩阵W是均匀随机选择的，因此每个类别的预测概率是均匀分布，并且等于1/10，其中10是类别数。 因此，每个示例的交叉熵是-log（0.1），应等于损失。

In [4]:

```
# 完成softmax_loss_naive，并实现使用嵌套循环的梯度的版本(naive)。
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 0.0)

# 就像SVM那样，请使用数值梯度检查作为调试工具。
# 数值梯度应接近分析梯度。
from daseCV.gradient_check import grad_check_sparse
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 0.0)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)

# 与SVM情况类似，使用正则化进行另一个梯度检查
loss, grad = softmax_loss_naive(W, X_dev, y_dev, 5e1)
f = lambda w: softmax_loss_naive(w, X_dev, y_dev, 5e1)[0]
grad_numerical = grad_check_sparse(f, W, grad, 10)
```

```
numerical: -1.329283 analytic: -1.329283, relative error: 4.246219e-08
numerical: -2.069483 analytic: -2.069483, relative error: 7.162094e-09
numerical: -0.252397 analytic: -0.252397, relative error: 1.907719e-07
numerical: -2.587240 analytic: -2.587240, relative error: 7.938916e-09
numerical: -2.471031 analytic: -2.471031, relative error: 2.525471e-08
numerical: 0.815281 analytic: 0.815281, relative error: 1.096930e-07
numerical: -1.625533 analytic: -1.625533, relative error: 1.537782e-08
numerical: -1.672472 analytic: -1.672472, relative error: 2.071797e-09
numerical: -1.371480 analytic: -1.371480, relative error: 1.919050e-08
numerical: 1.546283 analytic: 1.546283, relative error: 2.818494e-08
numerical: -3.316076 analytic: -3.316076, relative error: 3.746307e-09
numerical: 2.705743 analytic: 2.705742, relative error: 3.687880e-08
numerical: -2.073291 analytic: -2.073291, relative error: 3.641486e-08
numerical: 1.600752 analytic: 1.600752, relative error: 9.692682e-09
numerical: -3.497092 analytic: -3.497092, relative error: 3.494441e-09
numerical: -1.602276 analytic: -1.602276, relative error: 2.261423e-08
numerical: -0.657978 analytic: -0.657978, relative error: 1.360731e-07
numerical: 0.635344 analytic: 0.635344, relative error: 4.472060e-08
numerical: 4.375184 analytic: 4.375184, relative error: 5.581858e-09
numerical: 1.039570 analytic: 1.039570, relative error: 6.809016e-08
```

In [5]:

```
# 现在，我们有了softmax损失函数及其梯度的简单实现，
# 接下来要在 softmax_loss_vectorized 中完成一个向量化版本.
# 这两个版本应计算出相同的结果，但矢量化版本应更快。
tic = time.time()
loss_naive, grad_naive = softmax_loss_naive(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('naive loss: %e computed in %fs' % (loss_naive, toc - tic))

from daseCV.classifiers.softmax import softmax_loss_vectorized
tic = time.time()
loss_vectorized, grad_vectorized = softmax_loss_vectorized(W, X_dev, y_dev, 0.000005)
toc = time.time()
print('vectorized loss: %e computed in %fs' % (loss_vectorized, toc - tic))
```

```
# 正如前面在SVM练习中所做的一样，我们使用Frobenius范数比较两个版本梯度。
grad_difference = np.linalg.norm(grad_naive - grad_vectorized, ord='fro')
print('Loss difference: %f' % np.abs(loss_naive - loss_vectorized))
print('Gradient difference: %f' % grad_difference)
```

```
naive loss: 2.340605e+00 computed in 1.792801s
vectorized loss: 2.340605e+00 computed in 0.005356s
Loss difference: 0.000000
Gradient difference: 0.000000
```

In [6]:

```
# 使用验证集调整超参数（正则化强度和学习率）。您应该尝试不同的学习率和正则化强度范围；
# 如果您小心的话，您应该能够在验证集上获得超过0.35的精度。
from daseCV.classifiers import Softmax
results = {}
best_val = -1
best_softmax = None
learning_rates = np.random.randint(1,100,5) * 1e-8
regularization_strengths = np.random.randint(1,100,5) * 1e3

################################################################################
# 需要完成的事:
# 对验证集设置学习率和正则化强度。
# 这与之前SVM中做的类似;
# 保存训练效果最好的softmax分类器到best_softmax中。
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

for lr in learning_rates:
      for reg in regularization_strengths:
        cur_softmax = Softmax()
        loss_history = cur_softmax.train(X_train, y_train, learning_rate=lr, reg=reg,
        train_pred = cur_softmax.predict(X_train)
        val_pred = cur_softmax.predict(X_val)
        train_accuracy = np.sum(train_pred==y_train) / len(train_pred)
        val_accuracy = np.sum(val_pred==y_val) / len(val_pred)
        results[(lr, reg)] = (train_accuracy, val_accuracy)
        if val_accuracy > best_val:
            best_val = val_accuracy
            best_softmax = cur_softmax


# *****END OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****

# Print out results.
for lr, reg in sorted(results):
    train_accuracy, val_accuracy = results[(lr, reg)]
    print('lr %e reg %e train accuracy: %f val accuracy: %f' % (
                lr, reg, train_accuracy, val_accuracy))

print('best validation accuracy achieved during cross-validation: %f' % best_val)
```

```
lr 3.000000e-08 reg 6.000000e+03 train accuracy: 0.179918 val accuracy: 0.198000
lr 3.000000e-08 reg 8.000000e+03 train accuracy: 0.178898 val accuracy: 0.210000
lr 3.000000e-08 reg 4.000000e+04 train accuracy: 0.243837 val accuracy: 0.259000
lr 3.000000e-08 reg 6.700000e+04 train accuracy: 0.264816 val accuracy: 0.288000
lr 3.500000e-07 reg 6.000000e+03 train accuracy: 0.375510 val accuracy: 0.409000
lr 3.500000e-07 reg 8.000000e+03 train accuracy: 0.376612 val accuracy: 0.401000
lr 3.500000e-07 reg 4.000000e+04 train accuracy: 0.331204 val accuracy: 0.345000
lr 3.500000e-07 reg 6.700000e+04 train accuracy: 0.312551 val accuracy: 0.339000
lr 4.000000e-07 reg 6.000000e+03 train accuracy: 0.375469 val accuracy: 0.380000
lr 4.000000e-07 reg 8.000000e+03 train accuracy: 0.377776 val accuracy: 0.378000
lr 4.000000e-07 reg 4.000000e+04 train accuracy: 0.335143 val accuracy: 0.346000
lr 4.000000e-07 reg 6.700000e+04 train accuracy: 0.298694 val accuracy: 0.320000
lr 7.800000e-07 reg 6.000000e+03 train accuracy: 0.384633 val accuracy: 0.393000
lr 7.800000e-07 reg 8.000000e+03 train accuracy: 0.374041 val accuracy: 0.383000
```

```
lr 7.800000e-07 reg 4.000000e+04 train accuracy: 0.333367 val accuracy: 0.336000
lr 7.800000e-07 reg 6.700000e+04 train accuracy: 0.307633 val accuracy: 0.320000
lr 9.300000e-07 reg 6.000000e+03 train accuracy: 0.382286 val accuracy: 0.384000
lr 9.300000e-07 reg 8.000000e+03 train accuracy: 0.371776 val accuracy: 0.375000
lr 9.300000e-07 reg 4.000000e+04 train accuracy: 0.328612 val accuracy: 0.343000
lr 9.300000e-07 reg 6.700000e+04 train accuracy: 0.313327 val accuracy: 0.325000
best validation accuracy achieved during cross-validation: 0.409000
```

In [7]:

```
# 在测试集上评估
# 在测试集上评估最好的softmax
y_test_pred = best_softmax.predict(X_test)
test_accuracy = np.mean(y_test == y_test_pred)
print('softmax on raw pixels final test set accuracy: %f' % (test_accuracy, ))
```

```
softmax on raw pixels final test set accuracy: 0.377000
```

问题 2 - *对或错*

假设总训练损失定义为所有训练样本中每个数据点损失的总和。可能会有新的数据点添加到训练集中，同时SVM损失保持不变，但是对于Softmax分类器的损失而言，情况并非如此。

你的回答：对

你的解释：

对于svm分类器而言，有max操作允许部分分数对结果(s_yi-s_j<-1>)没有影响，softmax分类器会考虑到所有的数据的情况，因此对于softmax分类器，会产生损失的变化。softmax值都大于0的怎么加新点都会变大。通常情况下Softmax loss都会改变，因为无法保证新加入的数据的预测正确的概率达到1，当新加入输入预测正确的概率不等于1时就会引入Softmax loss。对SVM来讲是可以做到不引入新的损失的，只要标签类得分大于非标签类得分且它们之间的距离大于1。

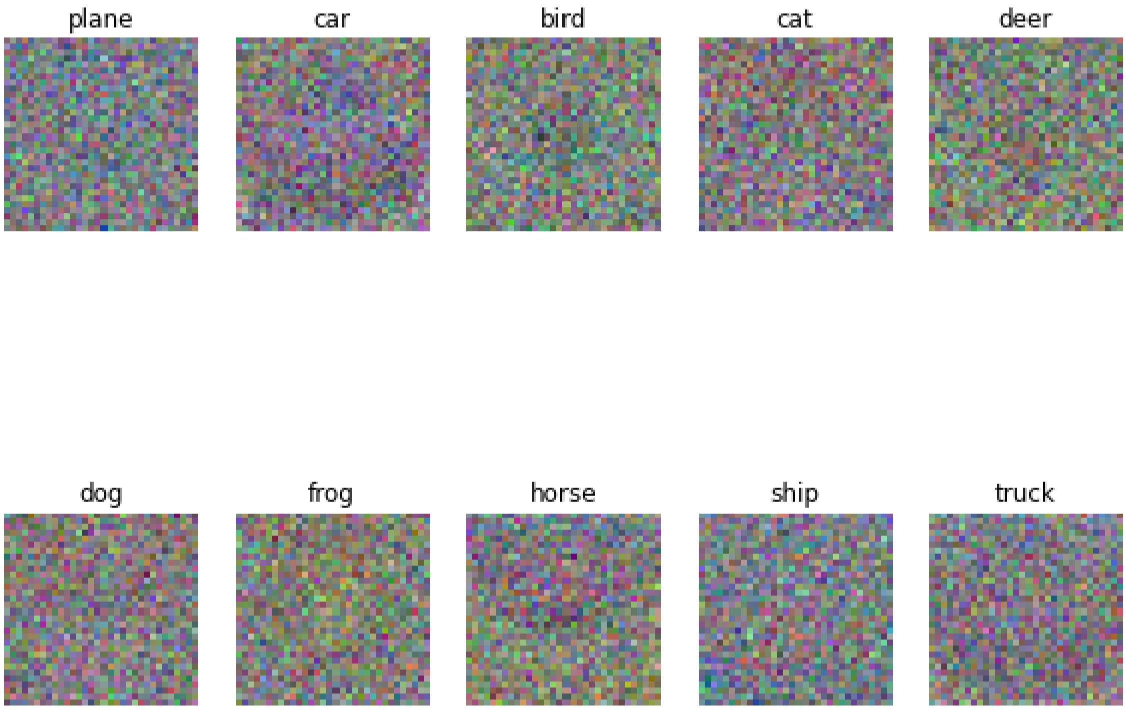
In [8]:

```
# 可视化每个类别的学习到的权重
w = best_softmax.W[:-1,:] # strip out the bias
w = w.reshape(32, 32, 3, 10)

w_min, w_max = np.min(w), np.max(w)

classes = ['plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'tru
for i in range(10):
    plt.subplot(2, 5, i + 1)

    # Rescale the weights to be between 0 and 255
    wimg = 255.0 * (w[:, :, :, i].squeeze() - w_min) / (w_max - w_min)
    plt.imshow(wimg.astype('uint8'))
    plt.axis('off')
    plt.title(classes[i])
```



Data for leaderboard

这里额外提供了一组未给标签的测试集X，用于leaderborad上的竞赛。

提示：该题的目的是鼓励同学们探索能够提升模型性能的方法。

In [9]:

```
# leaderboard的测试数据
X = np.load("./input/X_3073.npy")
################################################################################
# 需要完成的事情:
# 找到更合适的softmax
# 提示: 如果你不想花时间，你也可以直接使用上面已经训练好的best_softmax。
################################################################################
# *****START OF YOUR CODE (DO NOT DELETE/MODIFY THIS LINE)*****
softmax_leaderboard = best_softmax
preds = softmax_leaderboard.predict(X)
```

提醒：运行完下面代码之后，点击下面的submit，然后去leaderboard上查看你的成绩。本模型对应的成绩在phase3的leaderboard中。

In [10]:

```
import os
#输出格式
def output_file(preds, phase_id=3):
    path=os.getcwd()
    if not os.path.exists(path + '/output/phase_{}'.format(phase_id)):
        os.mkdir(path + '/output/phase_{}'.format(phase_id))
    path=path + '/output/phase_{}/prediction.npy'.format(phase_id)
    np.save(path,preds)
def zip_fun(phase_id=3):
    path=os.getcwd()
    output_path = path + '/output'
    files = os.listdir(output_path)
    for _file in files:
        if _file.find('zip') != -1:
            os.remove(output_path + '/' + _file)
    newpath=path+'/output/phase_{}'.format(phase_id)
```

```
    os.chdir(newpath)
    cmd = 'zip ../prediction_phase_{}.zip prediction.npy'.format(phase_id)
    os.system(cmd)
    os.chdir(path)
output_file(preds)
zip_fun()
```

In [ ]:

```

```