softmax

2021 年 4 月 8 日

[ ]:
```python
from google.colab import drive

drive.mount('/content/drive', force_remount=True)

# 输入 daseCV 所在的路径
# 'daseCV' 文件夹包括 '.py', 'classifiers' 和'datasets'文件夹
# 例如 'CV/assignments/assignment1/daseCV/'
FOLDERNAME = 'CV/assignments/assignment1/daseCV/'

assert FOLDERNAME is not None, "[!] Enter the foldername."

%cd drive/My\ Drive
%cp -r $FOLDERNAME ../../
%cd ../../
%cd daseCV/datasets/
!bash get_datasets.sh
%cd ../../
```

```
Mounted at /content/drive
/content/drive/My Drive
/content
/content/daseCV/datasets
--2021-04-03 12:56:42--  http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
Resolving www.cs.toronto.edu (www.cs.toronto.edu)… 128.100.3.30
Connecting to www.cs.toronto.edu (www.cs.toronto.edu)|128.100.3.30|:80…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 170498071 (163M) [application/x-gzip]
```

```
Saving to: ‘cifar-10-python.tar.gz’

cifar-10-python.tar 100%[===================>] 162.60M  70.4MB/s    in 2.3s

2021-04-03 12:56:45 (70.4 MB/s) - ‘cifar-10-python.tar.gz’ saved
[170498071/170498071]

cifar-10-batches-py/
cifar-10-batches-py/data_batch_4
cifar-10-batches-py/readme.html
cifar-10-batches-py/test_batch
cifar-10-batches-py/data_batch_3
cifar-10-batches-py/batches.meta
cifar-10-batches-py/data_batch_2
cifar-10-batches-py/data_batch_5
cifar-10-batches-py/data_batch_1
/content
```

1 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

[ ]:
```
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
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[ ]:
```
def get_CIFAR10_data(num_training=49000, num_validation=1000, num_test=1000,␣
↪num_dev=500):
    """
    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 memory issue)
    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,)
```

1.1 Softmax 分类器

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

[ ]:
```
# 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: 2.489394
sanity check: 2.302585
```

[ ]:
```
W[1:4]
```

[ ]:
```
array([[-5.63857262e-05,  1.42104178e-04, -1.63497641e-04,
         5.21989270e-05, -1.01614662e-04,  3.17728984e-05,
         1.08077893e-04, -2.56450380e-05, -2.30000045e-04,
         1.67030872e-05],
       [ 3.25436775e-05,  7.71187498e-06, -6.35514754e-05,
         2.31037970e-05, -3.83040886e-05, -1.77027887e-04,
         2.10226448e-05,  2.68918820e-05, -1.94214241e-05,
        -2.90174960e-05],
       [ 5.37011753e-05, -1.60069226e-04,  8.96287286e-05,
         1.82900324e-04,  1.85280903e-04, -1.11433733e-04,
        -1.67590517e-05, -7.13481146e-05,  4.07533752e-05,
         3.76939953e-05]])
```

问题 1

为什么我们期望损失接近-log（0.1)？简要说明。

答：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。

[ ]:
```
# 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: 1.957841 analytic: 1.957841, relative error: 6.166928e-08
numerical: -5.867751 analytic: -5.867751, relative error: 3.358170e-09
numerical: -0.634055 analytic: -0.634055, relative error: 1.251246e-08
numerical: 3.583884 analytic: 3.583884, relative error: 1.469139e-08
numerical: -0.694063 analytic: -0.694063, relative error: 5.483703e-08
numerical: 0.402959 analytic: 0.402959, relative error: 4.802526e-08
numerical: 0.306422 analytic: 0.306422, relative error: 4.179119e-08
numerical: -0.759939 analytic: -0.759940, relative error: 9.215787e-08
numerical: -1.202137 analytic: -1.202137, relative error: 2.961540e-08
numerical: 0.838005 analytic: 0.838005, relative error: 7.304510e-09
numerical: 0.499000 analytic: 0.499000, relative error: 2.633208e-08
numerical: 1.361596 analytic: 1.361596, relative error: 3.284206e-08
numerical: -3.165223 analytic: -3.165223, relative error: 1.507749e-08
numerical: 4.474712 analytic: 4.474712, relative error: 1.334087e-08
numerical: -0.208245 analytic: -0.208245, relative error: 2.036710e-07
numerical: 0.459994 analytic: 0.459994, relative error: 3.246983e-08
numerical: 0.946844 analytic: 0.946844, relative error: 4.074694e-08
numerical: 0.986594 analytic: 0.986594, relative error: 3.401871e-08
numerical: -2.064113 analytic: -2.064113, relative error: 1.278162e-08
numerical: -0.691414 analytic: -0.691415, relative error: 1.335196e-07
```

[ ]:
```
# 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: 2.489394e+00 computed in 0.582451s
vectorized loss: 2.489394e+00 computed in 0.019241s
Loss difference: 0.000000
Gradient difference: 0.000000
```

[ ]:
```
# 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 = [1e-7, 2e-7, 5e-7]
#regularization_strengths = [5e4, 1e8]
regularization_strengths =[(1+0.1*i)*1e4 for i in range(-3,4)] + [(5+0.1*i)*1e4␣
↪for i in range(-3,4)]

################################################################################
# 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 rs in regularization_strengths:
        softmax = Softmax()
        softmax.train(X_train, y_train, lr, rs, num_iters=2000)
        y_train_pred = softmax.predict(X_train)
        train_accuracy = np.mean(y_train == y_train_pred)
        y_val_pred = softmax.predict(X_val)
        val_accuracy = np.mean(y_val == y_val_pred)
        if val_accuracy > best_val:
            best_val = val_accuracy
            best_softmax = softmax
        results[(lr,rs)] = train_accuracy, val_accuracy

# *****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 1.000000e-07 reg 7.000000e+03 train accuracy: 0.339367 val accuracy: 0.344000
lr 1.000000e-07 reg 8.000000e+03 train accuracy: 0.350000 val accuracy: 0.335000
lr 1.000000e-07 reg 9.000000e+03 train accuracy: 0.356163 val accuracy: 0.363000
lr 1.000000e-07 reg 1.000000e+04 train accuracy: 0.359980 val accuracy: 0.358000
lr 1.000000e-07 reg 1.100000e+04 train accuracy: 0.358694 val accuracy: 0.364000
lr 1.000000e-07 reg 1.200000e+04 train accuracy: 0.358816 val accuracy: 0.367000
lr 1.000000e-07 reg 1.300000e+04 train accuracy: 0.364306 val accuracy: 0.367000
lr 1.000000e-07 reg 4.700000e+04 train accuracy: 0.331286 val accuracy: 0.345000
lr 1.000000e-07 reg 4.800000e+04 train accuracy: 0.331898 val accuracy: 0.352000
lr 1.000000e-07 reg 4.900000e+04 train accuracy: 0.333061 val accuracy: 0.359000
lr 1.000000e-07 reg 5.000000e+04 train accuracy: 0.328306 val accuracy: 0.338000
lr 1.000000e-07 reg 5.100000e+04 train accuracy: 0.328204 val accuracy: 0.348000
```

```
lr 1.000000e-07 reg 5.200000e+04 train accuracy: 0.327612 val accuracy: 0.347000
lr 1.000000e-07 reg 5.300000e+04 train accuracy: 0.321837 val accuracy: 0.339000
lr 2.000000e-07 reg 7.000000e+03 train accuracy: 0.377163 val accuracy: 0.395000
lr 2.000000e-07 reg 8.000000e+03 train accuracy: 0.377694 val accuracy: 0.388000
lr 2.000000e-07 reg 9.000000e+03 train accuracy: 0.374469 val accuracy: 0.386000
lr 2.000000e-07 reg 1.000000e+04 train accuracy: 0.371918 val accuracy: 0.394000
lr 2.000000e-07 reg 1.100000e+04 train accuracy: 0.371776 val accuracy: 0.386000
lr 2.000000e-07 reg 1.200000e+04 train accuracy: 0.368755 val accuracy: 0.385000
lr 2.000000e-07 reg 1.300000e+04 train accuracy: 0.371122 val accuracy: 0.382000
lr 2.000000e-07 reg 4.700000e+04 train accuracy: 0.331286 val accuracy: 0.351000
lr 2.000000e-07 reg 4.800000e+04 train accuracy: 0.325878 val accuracy: 0.336000
lr 2.000000e-07 reg 4.900000e+04 train accuracy: 0.332143 val accuracy: 0.347000
lr 2.000000e-07 reg 5.000000e+04 train accuracy: 0.331612 val accuracy: 0.341000
lr 2.000000e-07 reg 5.100000e+04 train accuracy: 0.323061 val accuracy: 0.338000
lr 2.000000e-07 reg 5.200000e+04 train accuracy: 0.327286 val accuracy: 0.340000
lr 2.000000e-07 reg 5.300000e+04 train accuracy: 0.321673 val accuracy: 0.339000
lr 5.000000e-07 reg 7.000000e+03 train accuracy: 0.379857 val accuracy: 0.388000
lr 5.000000e-07 reg 8.000000e+03 train accuracy: 0.378265 val accuracy: 0.391000
lr 5.000000e-07 reg 9.000000e+03 train accuracy: 0.374265 val accuracy: 0.378000
lr 5.000000e-07 reg 1.000000e+04 train accuracy: 0.369388 val accuracy: 0.371000
lr 5.000000e-07 reg 1.100000e+04 train accuracy: 0.372449 val accuracy: 0.385000
lr 5.000000e-07 reg 1.200000e+04 train accuracy: 0.366245 val accuracy: 0.370000
lr 5.000000e-07 reg 1.300000e+04 train accuracy: 0.363592 val accuracy: 0.379000
lr 5.000000e-07 reg 4.700000e+04 train accuracy: 0.332469 val accuracy: 0.345000
lr 5.000000e-07 reg 4.800000e+04 train accuracy: 0.325755 val accuracy: 0.341000
lr 5.000000e-07 reg 4.900000e+04 train accuracy: 0.330694 val accuracy: 0.335000
lr 5.000000e-07 reg 5.000000e+04 train accuracy: 0.330122 val accuracy: 0.337000
lr 5.000000e-07 reg 5.100000e+04 train accuracy: 0.329408 val accuracy: 0.339000
lr 5.000000e-07 reg 5.200000e+04 train accuracy: 0.327898 val accuracy: 0.342000
lr 5.000000e-07 reg 5.300000e+04 train accuracy: 0.320122 val accuracy: 0.337000
best validation accuracy achieved during cross-validation: 0.395000
```

[ ]:
```
print("softmax # 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,␣
↪))on raw pixels final test set accuracy: 0.369000")
```

```
softmax on raw pixels final test set accuracy: 0.369000
```

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 :对

你 :softmax 值都大于 0 的怎么加新点都会变大

[ ]:
```
# 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', 'truck']
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])
```

---

2 重要

这里是作业的结尾处，请执行以下步骤:

1. 点击 `File -> Save` 或者用 `control+s` 组合键，确保你最新的的 notebook 的作业已经保存到谷歌云。

2. 执行以下代码确保 .py 文件保存回你的谷歌云。

[ ]:
```
import os

FOLDER_TO_SAVE = os.path.join('drive/My Drive/', FOLDERNAME)
FILES_TO_SAVE = ['daseCV/classifiers/softmax.py']

for files in FILES_TO_SAVE:
  with open(os.path.join(FOLDER_TO_SAVE, '/'.join(files.split('/')[1:])), 'w') as f:
    f.write(''.join(open(files).readlines()))
```