ConvolutionalNetworks

2021 年 5 月 14 日

1　卷积网络

到目前为止，我们已经成功使用深层全连接网络，并使用它们来探索不同的优化策略和网络结构。全连接网络是很好的实验平台，因为它们的计算效率很高，但实际上，所有最新结果都使用卷积网络。

首先，你将实现几个在卷积网络中使用的层类型。然后，您将使用这些层在 CIFAR-10 数据集上训练卷积网络。

```
[5]:  # As usual, a bit of setup
      import numpy as np
      import matplotlib.pyplot as plt
      from daseCV.classifiers.cnn import *
      from daseCV.data_utils import get_CIFAR10_data
      from daseCV.gradient_check import eval_numerical_gradient_array,␣
      ↪eval_numerical_gradient
      from daseCV.layers import *
      from daseCV.fast_layers import *
      from daseCV.solver import Solver


      %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 external modules
      # see http://stackoverflow.com/questions/1907993/
      ↪autoreload-of-modules-in-ipython
```

```
%load_ext autoreload
%autoreload 2


def rel_error(x, y):
  """ returns relative error """
  return np.max(np.abs(x - y) / (np.maximum(1e-8, np.abs(x) + np.abs(y))))
```

run the following from the daseCV directory and try again:

python setup.py build_ext --inplace

You may also need to restart your iPython kernel

[12]:
```
!pwd
```

/content

[11]:
```
!cd daseCV/ && python setup.py build_ext --inplace
```

Compiling im2col_cython.pyx because it changed.

[1/1] Cythonizing im2col_cython.pyx

/usr/local/lib/python3.7/dist-packages/Cython/Compiler/Main.py:369:

FutureWarning: Cython directive 'language_level' not set, using 2 for now (Py2).

This will change in a later release! File: /content/daseCV/im2col_cython.pyx

  tree = Parsing.p_module(s, pxd, full_module_name)

running build_ext

building 'im2col_cython' extension

creating build

creating build/temp.linux-x86_64-3.7

x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g

-fwrapv -O2 -Wall -g -fdebug-prefix-

map=/build/python3.7-a56wZI/python3.7-3.7.10=. -fstack-protector-strong -Wformat

-Werror=format-security -g -fdebug-prefix-

map=/build/python3.7-a56wZI/python3.7-3.7.10=. -fstack-protector-strong -Wformat

-Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2 -fPIC

-I/usr/local/lib/python3.7/dist-packages/numpy/core/include

-I/usr/include/python3.7m -c im2col_cython.c -o

build/temp.linux-x86_64-3.7/im2col_cython.o

In file included from /usr/local/lib/python3.7/dist-

packages/numpy/core/include/numpy/ndarraytypes.h:1822:0,

                 from /usr/local/lib/python3.7/dist-

packages/numpy/core/include/numpy/ndarrayobject.h:12,

                 from /usr/local/lib/python3.7/dist-

packages/numpy/core/include/numpy/arrayobject.h:4,

                 from im2col_cython.c:629:
/usr/local/lib/python3.7/dist-

packages/numpy/core/include/numpy/npy_1_7_deprecated_api.h:17:2:
warning: #warning "Using deprecated NumPy API, disable it with
" "#define NPY_NO_DEPRECATED_API NPY_1_7_API_VERSION" [-Wcpp]
 #warning "Using deprecated NumPy API, disable it with " \
  ^~~~~~~
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions
-Wl,-Bsymbolic-functions -Wl,-z,relro -Wl,-Bsymbolic-functions -Wl,-z,relro -g
-fdebug-prefix-map=/build/python3.7-a56wZI/python3.7-3.7.10=. -fstack-protector-
strong -Wformat -Werror=format-security -Wdate-time -D_FORTIFY_SOURCE=2
build/temp.linux-x86_64-3.7/im2col_cython.o -o
/content/daseCV/im2col_cython.cpython-37m-x86_64-linux-gnu.so

```
[13]: # Load the (preprocessed) CIFAR10 data.

      data = get_CIFAR10_data()
      for k, v in data.items():
        print('%s: ' % k, v.shape)
```

```
X_train:  (49000, 3, 32, 32)
y_train:  (49000,)
X_val:  (1000, 3, 32, 32)
y_val:  (1000,)
X_test:  (1000, 3, 32, 32)
y_test:  (1000,)
```

[ ]:

2 卷积：简单的正向传播

卷积网络的核心是卷积运算。在文件 daseCV/layers.py 中的函数 conv_forward_naive 里实现卷积层的正向传播。

此时，你不必太担心效率。只需以你最清楚的方式编写代码即可。

您可以通过运行以下 cell 来测试你的代码：

```
[8]: x_shape = (2, 3, 4, 4)
w_shape = (3, 3, 4, 4)
x = np.linspace(-0.1, 0.5, num=np.prod(x_shape)).reshape(x_shape)
w = np.linspace(-0.2, 0.3, num=np.prod(w_shape)).reshape(w_shape)
b = np.linspace(-0.1, 0.2, num=3)

conv_param = {'stride': 2, 'pad': 1}
out, _ = conv_forward_naive(x, w, b, conv_param)
correct_out = np.array([[[[-0.08759809, -0.10987781],
                          [-0.18387192, -0.2109216 ]],
                         [[ 0.21027089,  0.21661097],
                          [ 0.22847626,  0.23004637]],
                         [[ 0.50813986,  0.54309974],
                          [ 0.64082444,  0.67101435]]],
                        [[[-0.98053589, -1.03143541],
                          [-1.19128892, -1.24695841]],
                         [[ 0.69108355,  0.66880383],
                          [ 0.59480972,  0.56776003]],
                         [[ 2.36270298,  2.36904306],
                          [ 2.38090835,  2.38247847]]]])

# Compare your output to ours; difference should be around e-8
print('Testing conv_forward_naive')
print('difference: ', rel_error(out, correct_out))
```

```
Testing conv_forward_naive
difference:  2.2121476417505994e-08
```

```
[6]: A = np.arange(95,99).reshape(2,2)
```

```
[7]: np.pad(A,((3,2),(2,3)),'constant',constant_values = ((0,0),(1,2)))
```

```
[7]: array([[ 1,  1,  0,  0,  2,  2,  2],
            [ 1,  1,  0,  0,  2,  2,  2],
            [ 1,  1,  0,  0,  2,  2,  2],
            [ 1,  1, 95, 96,  2,  2,  2],
            [ 1,  1, 97, 98,  2,  2,  2],
            [ 1,  1,  0,  0,  2,  2,  2],
            [ 1,  1,  0,  0,  2,  2,  2]])
```

3   补充：通过卷积对进行图像处理

为了检查你的代码以及更好的理解卷积层可以实现的操作类型，我们将设置一个包含两个图像的输入，并手动设置执行常见图像处理操作（灰度转换和边缘检测）的滤镜。卷积的正向传播会将这些操作应用于每个输入图像。然后，我们可以将结果可视化以此检查准确性。

```
[15]: from imageio import imread
      from PIL import Image

      kitten = imread('notebook_images/kitten.jpg')
      puppy = imread('notebook_images/puppy.jpg')
      # kitten is wide, and puppy is already square
      d = kitten.shape[1] - kitten.shape[0]
      kitten_cropped = kitten[:, d//2:-d//2, :]

      img_size = 200    # Make this smaller if it runs too slow
      resized_puppy = np.array(Image.fromarray(puppy).resize((img_size, img_size)))
      resized_kitten = np.array(Image.fromarray(kitten_cropped).resize((img_size,
       ↪img_size)))
      x = np.zeros((2, 3, img_size, img_size))
      x[0, :, :, :] = resized_puppy.transpose((2, 0, 1))
      x[1, :, :, :] = resized_kitten.transpose((2, 0, 1))

      # Set up a convolutional weights holding 2 filters, each 3x3
      w = np.zeros((2, 3, 3, 3))
```

```python
# The first filter converts the image to grayscale.
# Set up the red, green, and blue channels of the filter.
w[0, 0, :, :] = [[0, 0, 0], [0, 0.3, 0], [0, 0, 0]]
w[0, 1, :, :] = [[0, 0, 0], [0, 0.6, 0], [0, 0, 0]]
w[0, 2, :, :] = [[0, 0, 0], [0, 0.1, 0], [0, 0, 0]]

# Second filter detects horizontal edges in the blue channel.
w[1, 2, :, :] = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]

# Vector of biases. We don't need any bias for the grayscale
# filter, but for the edge detection filter we want to add 128
# to each output so that nothing is negative.
b = np.array([0, 128])

# Compute the result of convolving each input in x with each filter in w,
# offsetting by b, and storing the results in out.
out, _ = conv_forward_naive(x, w, b, {'stride': 1, 'pad': 1})

def imshow_no_ax(img, normalize=True):
    """ Tiny helper to show images as uint8 and remove axis labels """
    if normalize:
        img_max, img_min = np.max(img), np.min(img)
        img = 255.0 * (img - img_min) / (img_max - img_min)
    plt.imshow(img.astype('uint8'))
    plt.gca().axis('off')

# Show the original images and the results of the conv operation
plt.subplot(2, 3, 1)
imshow_no_ax(puppy, normalize=False)
plt.title('Original image')
plt.subplot(2, 3, 2)
imshow_no_ax(out[0, 0])
plt.title('Grayscale')
plt.subplot(2, 3, 3)
imshow_no_ax(out[0, 1])
plt.title('Edges')
```
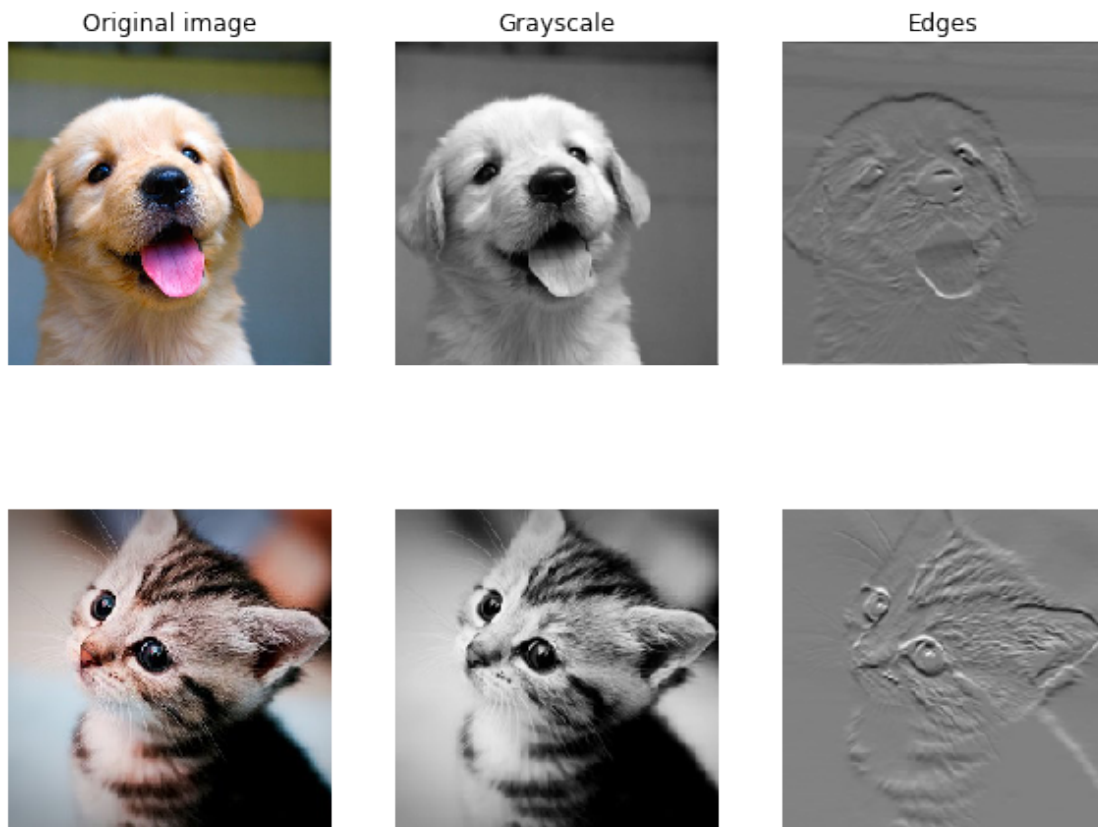
```
plt.subplot(2, 3, 4)
imshow_no_ax(kitten_cropped, normalize=False)
plt.subplot(2, 3, 5)
imshow_no_ax(out[1, 0])
plt.subplot(2, 3, 6)
imshow_no_ax(out[1, 1])
plt.show()
```


Original image   Grayscale   Edges

4　卷积：简单的反向传播

在文件 daseCV/layers.py 的 conv_backward_naive 函数中实现卷积操作的反向传播。同样，你不必太担心计算效率。

完成后，运行以下 cell 来检查你的反向传播的正确性。

```
[16]: np.random.seed(231)
x = np.random.randn(4, 3, 5, 5)
w = np.random.randn(2, 3, 3, 3)
b = np.random.randn(2,)
dout = np.random.randn(4, 2, 5, 5)
conv_param = {'stride': 1, 'pad': 1}

dx_num = eval_numerical_gradient_array(lambda x: conv_forward_naive(x, w, b,␣
 ↪conv_param)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: conv_forward_naive(x, w, b,␣
 ↪conv_param)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: conv_forward_naive(x, w, b,␣
 ↪conv_param)[0], b, dout)

out, cache = conv_forward_naive(x, w, b, conv_param)
dx, dw, db = conv_backward_naive(dout, cache)

# Your errors should be around e-8 or less.
print('Testing conv_backward_naive function')
print('dx error: ', rel_error(dx, dx_num))
print('dw error: ', rel_error(dw, dw_num))
print('db error: ', rel_error(db, db_num))
```

```
Testing conv_backward_naive function
dx error:  1.2152488219481614e-08
dw error:  1.434853118517851e-10
db error:  1.1987307048126593e-10
```

5   最大池化: 简单的正向传播

在文件 `daseCV/layers.py` 中的 `max_pool_forward_naive` 函数里实现最大池化操作的正向传播。同样，不必太担心计算效率。

通过运行以下 cell 检查你的代码:

```
[17]: x_shape = (2, 3, 4, 4)
      x = np.linspace(-0.3, 0.4, num=np.prod(x_shape)).reshape(x_shape)
      pool_param = {'pool_width': 2, 'pool_height': 2, 'stride': 2}


      out, _ = max_pool_forward_naive(x, pool_param)


      correct_out = np.array([[[[-0.26315789, -0.24842105],
                                [-0.20421053, -0.18947368]],
                               [[-0.14526316, -0.13052632],
                                [-0.08631579, -0.07157895]],
                               [[-0.02736842, -0.01263158],
                                [ 0.03157895,  0.04631579]]],
                              [[[ 0.09052632,  0.10526316],
                                [ 0.14947368,  0.16421053]],
                               [[ 0.20842105,  0.22315789],
                                [ 0.26736842,  0.28210526]],
                               [[ 0.32631579,  0.34105263],
                                [ 0.38526316,  0.4       ]]]])

      # Compare your output with ours. Difference should be on the order of e-8.
      print('Testing max_pool_forward_naive function:')
      print('difference: ', rel_error(out, correct_out))
```

```
Testing max_pool_forward_naive function:
difference:  4.1666665157267834e-08
```

6   最大池化: 简单的反向传播

在文件 `daseCV/layers.py` 中的 `max_pool_backward_naive` 函数里实现最大池化操作的反向传播。同样，不必太担心计算效率。

通过运行以下 cell 检查你的代码:

```
[18]: np.random.seed(231)
      x = np.random.randn(3, 2, 8, 8)
      dout = np.random.randn(3, 2, 4, 4)
      pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}
```

```
dx_num = eval_numerical_gradient_array(lambda x: max_pool_forward_naive(x,␣
 ↪pool_param)[0], x, dout)


out, cache = max_pool_forward_naive(x, pool_param)
dx = max_pool_backward_naive(dout, cache)


# Your error should be on the order of e-12
print('Testing max_pool_backward_naive function:')
print('dx error: ', rel_error(dx, dx_num))
```

```
Testing max_pool_backward_naive function:
dx error:  3.27562514223145e-12
```

7   Fast layers

让卷积和池化层更快可能有点难度。为了减轻你的痛苦，我们在文件 `daseCV/fast_layers.py` 中为卷积和池化层提供了正向和反向传播的快速版本。

快速卷积的实现依赖于 Cython 扩展。要编译它，你需要在 `daseCV` 目录中运行以下命令：

`python setup.py build_ext --inplace`

卷积和池化层的快速版本的 API 与你在之前实现的完全相同：正向传播接收数据、权重和参数，并产生输出和缓存对象；反向传播接收返回的导数和缓存对象，并针对数据和权重生成梯度。

提示： 只有当池化区域不重叠并对输入进行平铺时，池化的快速实现才能表现出最好的性能。如果不满足这些条件，那么快速池化将不会比原来的的实现快很多。

您可以通过运行以下代码和之前的版本之间进行性能的比较：

```
[15]: !cd daseCV/ && python setup.py build_ext --inplace
```

```
running build_ext
```

```
[17]: # Rel errors should be around e-9 or less
      from daseCV.fast_layers import conv_forward_fast, conv_backward_fast
      from time import time
      np.random.seed(231)
```

```python
x = np.random.randn(100, 3, 31, 31)
w = np.random.randn(25, 3, 3, 3)
b = np.random.randn(25,)
dout = np.random.randn(100, 25, 16, 16)
conv_param = {'stride': 2, 'pad': 1}

t0 = time()
out_naive, cache_naive = conv_forward_naive(x, w, b, conv_param)
t1 = time()
out_fast, cache_fast = conv_forward_fast(x, w, b, conv_param)
t2 = time()

print('Testing conv_forward_fast:')
print('Naive: %fs' % (t1 - t0))
print('Fast: %fs' % (t2 - t1))
print('Speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('Difference: ', rel_error(out_naive, out_fast))

t0 = time()
dx_naive, dw_naive, db_naive = conv_backward_naive(dout, cache_naive)
t1 = time()
dx_fast, dw_fast, db_fast = conv_backward_fast(dout, cache_fast)
t2 = time()

print('\nTesting conv_backward_fast:')
print('Naive: %fs' % (t1 - t0))
print('Fast: %fs' % (t2 - t1))
print('Speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('dx difference: ', rel_error(dx_naive, dx_fast))
print('dw difference: ', rel_error(dw_naive, dw_fast))
print('db difference: ', rel_error(db_naive, db_fast))
```

```
import successfully
Testing conv_forward_fast:
Naive: 1.606453s
Fast: 0.030559s
```

```
Speedup: 52.568782x
Difference:   3.941126281156453e-11


Testing conv_backward_fast:
Naive: 41.710783s
Fast: 0.014440s
Speedup: 2888.594172x
dx difference:   1.383704034070129e-11
dw difference:   9.671310178940481e-13
db difference:   3.1393858025571252e-15
```

```
[22]: # Relative errors should be close to 0.0
      from daseCV.fast_layers import max_pool_forward_fast, max_pool_backward_fast
      np.random.seed(231)
      x = np.random.randn(100, 3, 32, 32)
      dout = np.random.randn(100, 3, 16, 16)
      pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}

      t0 = time()
      out_naive, cache_naive = max_pool_forward_naive(x, pool_param)
      t1 = time()
      out_fast, cache_fast = max_pool_forward_fast(x, pool_param)
      t2 = time()

      print('Testing pool_forward_fast:')
      print('Naive: %fs' % (t1 - t0))
      print('fast: %fs' % (t2 - t1))
      print('speedup: %fx' % ((t1 - t0) / (t2 - t1)))
      print('difference: ', rel_error(out_naive, out_fast))

      t0 = time()
      dx_naive = max_pool_backward_naive(dout, cache_naive)
      t1 = time()
      dx_fast = max_pool_backward_fast(dout, cache_fast)
      t2 = time()

      print('\nTesting pool_backward_fast:')
```

```
print('Naive: %fs' % (t1 - t0))
print('fast: %fs' % (t2 - t1))
print('speedup: %fx' % ((t1 - t0) / (t2 - t1)))
print('dx difference: ', rel_error(dx_naive, dx_fast))
```

Testing pool_forward_fast:

Naive: 0.372830s

fast: 0.003782s

speedup: 98.591766x

difference: 0.0

Testing pool_backward_fast:

Naive: 1.183513s

fast: 0.014685s

speedup: 80.595126x

dx difference: 0.0

8   卷积 "sandwich" 层

之前，我们引入了 "sandwich" 层的概念，该层将多种操作组合成常用的模式。在文件 daseCV/layer_utils.py 中，您会找到一些实现卷积网络常用模式的 sandwich 层。运行下面的 cell 以检查它们是否正常工作。

```
[23]: from daseCV.layer_utils import conv_relu_pool_forward, conv_relu_pool_backward
      np.random.seed(231)
      x = np.random.randn(2, 3, 16, 16)
      w = np.random.randn(3, 3, 3, 3)
      b = np.random.randn(3,)
      dout = np.random.randn(2, 3, 8, 8)
      conv_param = {'stride': 1, 'pad': 1}
      pool_param = {'pool_height': 2, 'pool_width': 2, 'stride': 2}


      out, cache = conv_relu_pool_forward(x, w, b, conv_param, pool_param)
      dx, dw, db = conv_relu_pool_backward(dout, cache)
```

```
dx_num = eval_numerical_gradient_array(lambda x: conv_relu_pool_forward(x, w,␣
 ↪b, conv_param, pool_param)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: conv_relu_pool_forward(x, w,␣
 ↪b, conv_param, pool_param)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: conv_relu_pool_forward(x, w,␣
 ↪b, conv_param, pool_param)[0], b, dout)

# Relative errors should be around e-8 or less
print('Testing conv_relu_pool')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing conv_relu_pool
dx error:  9.591132621921372e-09
dw error:  5.802391137330214e-09
db error:  1.0146343411762047e-09
```

[24]:
```
from daseCV.layer_utils import conv_relu_forward, conv_relu_backward
np.random.seed(231)
x = np.random.randn(2, 3, 8, 8)
w = np.random.randn(3, 3, 3, 3)
b = np.random.randn(3,)
dout = np.random.randn(2, 3, 8, 8)
conv_param = {'stride': 1, 'pad': 1}

out, cache = conv_relu_forward(x, w, b, conv_param)
dx, dw, db = conv_relu_backward(dout, cache)

dx_num = eval_numerical_gradient_array(lambda x: conv_relu_forward(x, w, b,␣
 ↪conv_param)[0], x, dout)
dw_num = eval_numerical_gradient_array(lambda w: conv_relu_forward(x, w, b,␣
 ↪conv_param)[0], w, dout)
db_num = eval_numerical_gradient_array(lambda b: conv_relu_forward(x, w, b,␣
 ↪conv_param)[0], b, dout)
```

```
# Relative errors should be around e-8 or less
print('Testing conv_relu:')
print('dx error: ', rel_error(dx_num, dx))
print('dw error: ', rel_error(dw_num, dw))
print('db error: ', rel_error(db_num, db))
```

```
Testing conv_relu:
dx error:  1.5218619980349303e-09
dw error:  2.702022646099404e-10
db error:  1.451272393591721e-10
```

9 三层卷积网络

现在，你已经实现了所有必需的层，我们可以将它们组合成一个简单的卷积网络。

打开文件 daseCV/classifiers/cnn.py，并完成 ThreeLayerConvNet 类。请记住，您可以使用 fast/sandwich 层（以及提供给你）。运行以下 cell 以帮助你调试：

9.1 检查 loss

建立新网络后，您应该做的第一件事就是检查损失。当我们使用 softmax 损失时，对于 C 个类别我们期望随机权重的损失（没有正则化）大约为 log(C)。当我们添加正则化时，损失应该会略有增加。

```
[26]: model = ThreeLayerConvNet()


N = 50
X = np.random.randn(N, 3, 32, 32)
y = np.random.randint(10, size=N)


loss, grads = model.loss(X, y)
print('Initial loss (no regularization): ', loss)


model.reg = 0.5
loss, grads = model.loss(X, y)
print('Initial loss (with regularization): ', loss)
```

```
Initial loss (no regularization):  2.302585412176879
Initial loss (with regularization):  2.508542482169898
```

9.2 梯度检查

在损失看起来合理之后，请使用数值梯度检查来确保您的反向传播是正确的。使用数值梯度检查时，应在每一层使用少量的人工数据和少量的神经元。注意：正确的实现可能仍然会出现相对误差，最高可达 e-2。

```
[27]: num_inputs = 2
      #for testing bp, set input num to a small number.
      input_dim = (3, 16, 16)
      reg = 0.0
      num_classes = 10
      np.random.seed(231)
      X = np.random.randn(num_inputs, *input_dim)
      y = np.random.randint(num_classes, size=num_inputs)


      model = ThreeLayerConvNet(num_filters=3, filter_size=3,
                                input_dim=input_dim, hidden_dim=7,
                                dtype=np.float64)
      loss, grads = model.loss(X, y)
      # Errors should be small, but correct implementations may have
      # relative errors up to the order of e-2
      for param_name in sorted(grads):
          f = lambda _: model.loss(X, y)[0]
          param_grad_num = eval_numerical_gradient(f, model.params[param_name],␣
       ↪verbose=False, h=1e-6)
          e = rel_error(param_grad_num, grads[param_name])
          print('%s max relative error: %e' % (param_name, rel_error(param_grad_num,␣
       ↪grads[param_name])))
```

```
W1 max relative error: 1.380104e-04
W2 max relative error: 1.822723e-02
W3 max relative error: 3.064049e-04
b1 max relative error: 3.477652e-05
b2 max relative error: 2.516375e-03
```

```
b3 max relative error: 7.945660e-10
```

9.3 小样本的过拟合

一个不错的技巧是仅用少量训练样本来训练模型。您应该能够过度拟合较小的数据集，这将得到非常高的训练准确度和相对较低的验证准确度。

```
[28]: np.random.seed(231)

num_train = 100
small_data = {
  'X_train': data['X_train'][:num_train],
  'y_train': data['y_train'][:num_train],
  'X_val': data['X_val'],
  'y_val': data['y_val'],
}


model = ThreeLayerConvNet(weight_scale=1e-2)


solver = Solver(model, small_data,
                num_epochs=15, batch_size=50,
                update_rule='adam',
                optim_config={
                    'learning_rate': 1e-3,
                },
                verbose=True, print_every=1)
solver.train()
```

```
(Iteration 1 / 30) loss: 2.414060
(Epoch 0 / 15) train acc: 0.190000; val_acc: 0.124000
(Iteration 2 / 30) loss: 15.209270
(Epoch 1 / 15) train acc: 0.070000; val_acc: 0.102000
(Iteration 3 / 30) loss: 11.588512
(Iteration 4 / 30) loss: 4.783074
(Epoch 2 / 15) train acc: 0.080000; val_acc: 0.087000
(Iteration 5 / 30) loss: 6.527872
(Iteration 6 / 30) loss: 4.945386
```

(Epoch 3 / 15) train acc: 0.110000; val_acc: 0.106000
(Iteration 7 / 30) loss: 3.672949
(Iteration 8 / 30) loss: 3.458625
(Epoch 4 / 15) train acc: 0.170000; val_acc: 0.090000
(Iteration 9 / 30) loss: 2.185554
(Iteration 10 / 30) loss: 2.316694
(Epoch 5 / 15) train acc: 0.110000; val_acc: 0.074000
(Iteration 11 / 30) loss: 2.300313
(Iteration 12 / 30) loss: 2.301488
(Epoch 6 / 15) train acc: 0.110000; val_acc: 0.075000
(Iteration 13 / 30) loss: 2.297388
(Iteration 14 / 30) loss: 2.300745
(Epoch 7 / 15) train acc: 0.130000; val_acc: 0.080000
(Iteration 15 / 30) loss: 2.293541
(Iteration 16 / 30) loss: 2.294315
(Epoch 8 / 15) train acc: 0.130000; val_acc: 0.079000
(Iteration 17 / 30) loss: 2.300962
(Iteration 18 / 30) loss: 2.291868
(Epoch 9 / 15) train acc: 0.130000; val_acc: 0.079000
(Iteration 19 / 30) loss: 2.293962
(Iteration 20 / 30) loss: 2.297008
(Epoch 10 / 15) train acc: 0.130000; val_acc: 0.079000
(Iteration 21 / 30) loss: 2.288493
(Iteration 22 / 30) loss: 2.308738
(Epoch 11 / 15) train acc: 0.110000; val_acc: 0.076000
(Iteration 23 / 30) loss: 2.288925
(Iteration 24 / 30) loss: 2.285378
(Epoch 12 / 15) train acc: 0.110000; val_acc: 0.076000
(Iteration 25 / 30) loss: 2.299431
(Iteration 26 / 30) loss: 2.286134
(Epoch 13 / 15) train acc: 0.130000; val_acc: 0.080000
(Iteration 27 / 30) loss: 2.286441
(Iteration 28 / 30) loss: 2.281284
(Epoch 14 / 15) train acc: 0.130000; val_acc: 0.080000
(Iteration 29 / 30) loss: 2.293451
(Iteration 30 / 30) loss: 2.294649
(Epoch 15 / 15) train acc: 0.130000; val_acc: 0.080000

Plotting the loss, training accuracy, and validation accuracy should show clear overfitting:

```
[29]: plt.subplot(2, 1, 1)
plt.plot(solver.loss_history, 'o')
plt.xlabel('iteration')
plt.ylabel('loss')

plt.subplot(2, 1, 2)
plt.plot(solver.train_acc_history, '-o')
plt.plot(solver.val_acc_history, '-o')
plt.legend(['train', 'val'], loc='upper left')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.show()
```
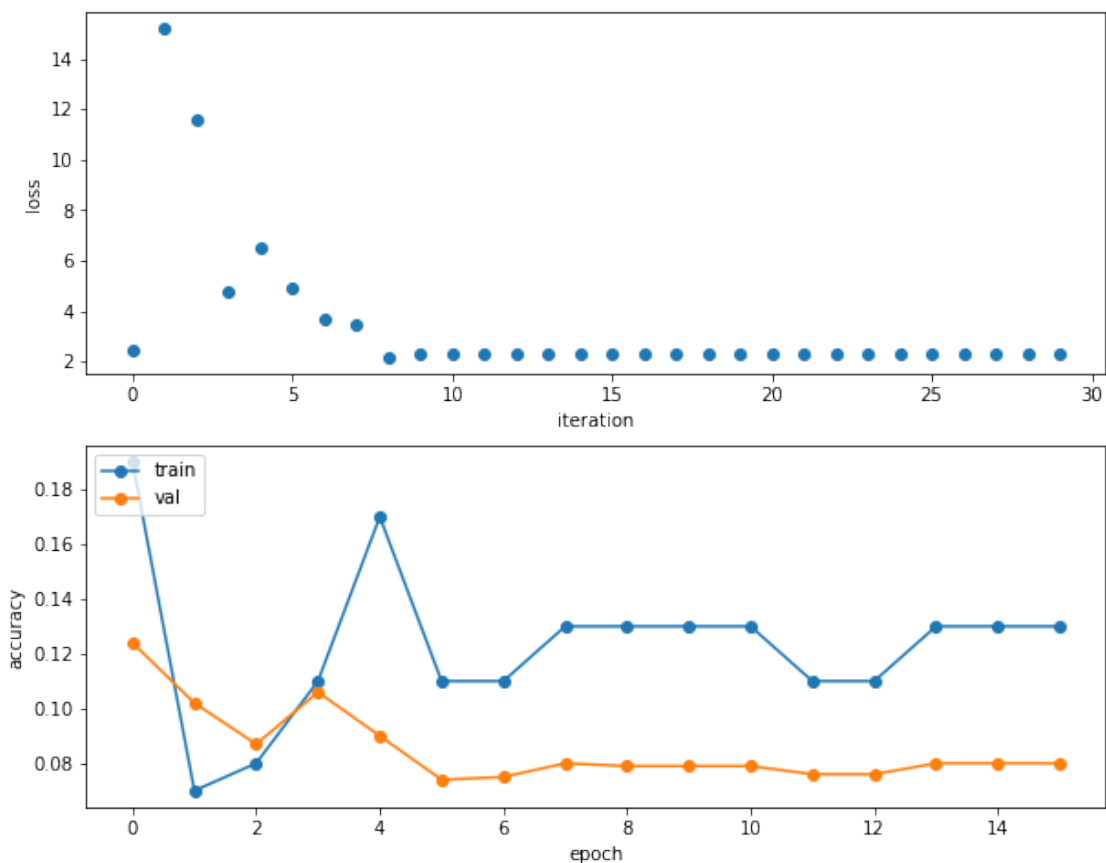
9.4　训练网络

将三层卷积网络训练一个 epoch，在训练集上将达到 40% 以上的准确度：

```
[30]: model = ThreeLayerConvNet(weight_scale=0.001, hidden_dim=500, reg=0.001)


      solver = Solver(model, data,
                      num_epochs=1, batch_size=50,
                      update_rule='adam',
                      optim_config={
                          'learning_rate': 1e-3,
                      },
                      verbose=True, print_every=20)
      solver.train()
```

```
(Iteration 1 / 980) loss: 2.304740
(Epoch 0 / 1) train acc: 0.103000; val_acc: 0.107000
(Iteration 21 / 980) loss: 3.846877
(Iteration 41 / 980) loss: 4.704070
(Iteration 61 / 980) loss: 4.507599
(Iteration 81 / 980) loss: 4.519320
(Iteration 101 / 980) loss: 4.513614
(Iteration 121 / 980) loss: 4.518971
(Iteration 141 / 980) loss: 4.516508
(Iteration 161 / 980) loss: 4.516718
(Iteration 181 / 980) loss: 4.514120
(Iteration 201 / 980) loss: 4.521260
(Iteration 221 / 980) loss: 4.524075
(Iteration 241 / 980) loss: 4.528673
(Iteration 261 / 980) loss: 4.528635
(Iteration 281 / 980) loss: 4.533946
(Iteration 301 / 980) loss: 4.527037
(Iteration 321 / 980) loss: 4.528690
(Iteration 341 / 980) loss: 4.540102
(Iteration 361 / 980) loss: 4.533069
(Iteration 381 / 980) loss: 4.532222
(Iteration 401 / 980) loss: 4.531955
```

```
(Iteration 421 / 980) loss: 4.534038
(Iteration 441 / 980) loss: 4.532000
(Iteration 461 / 980) loss: 4.537123
(Iteration 481 / 980) loss: 4.531179
(Iteration 501 / 980) loss: 4.534809
(Iteration 521 / 980) loss: 4.540142
(Iteration 541 / 980) loss: 4.531800
(Iteration 561 / 980) loss: 4.534122
(Iteration 581 / 980) loss: 4.530476
(Iteration 601 / 980) loss: 4.535810
(Iteration 621 / 980) loss: 4.534640
(Iteration 641 / 980) loss: 4.535525
(Iteration 661 / 980) loss: 4.533861
(Iteration 681 / 980) loss: 4.533796
(Iteration 701 / 980) loss: 4.533728
(Iteration 721 / 980) loss: 4.533682
(Iteration 741 / 980) loss: 4.534943
(Iteration 761 / 980) loss: 4.533825
(Iteration 781 / 980) loss: 4.535171
(Iteration 801 / 980) loss: 4.534185
(Iteration 821 / 980) loss: 4.535395
(Iteration 841 / 980) loss: 4.533331
(Iteration 861 / 980) loss: 4.534652
(Iteration 881 / 980) loss: 4.535890
(Iteration 901 / 980) loss: 4.538848
(Iteration 921 / 980) loss: 4.538817
(Iteration 941 / 980) loss: 4.536487
(Iteration 961 / 980) loss: 4.541288
(Epoch 1 / 1) train acc: 0.118000; val_acc: 0.112000
```

9.5 可视化过滤器

You can visualize the first-layer convolutional filters from the trained network by running the following: 您可以通过运行以下命令可视化训练好的第一层卷积过滤器：

```
[31]: from daseCV.vis_utils import visualize_grid

      grid = visualize_grid(model.params['W1'].transpose(0, 2, 3, 1))
      plt.imshow(grid.astype('uint8'))
      plt.axis('off')
      plt.gcf().set_size_inches(5, 5)
      plt.show()
```
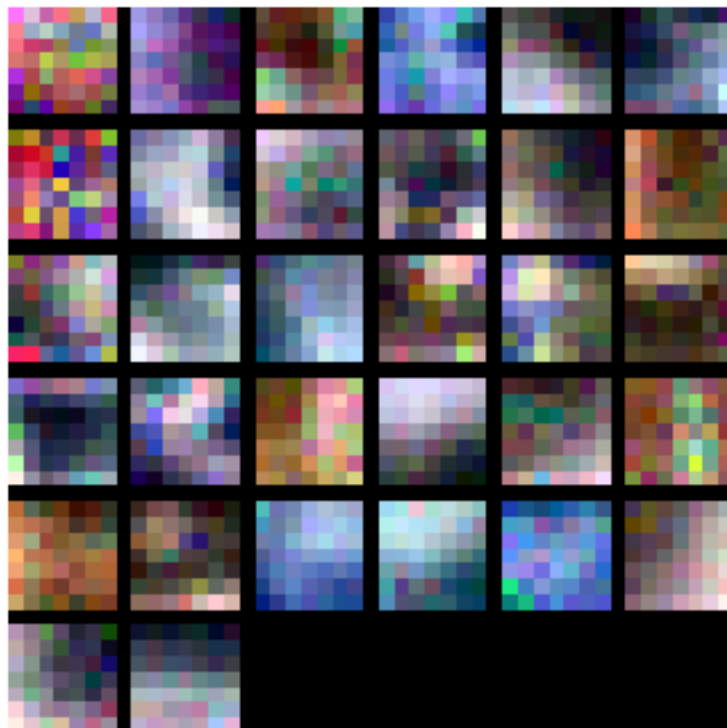

10   空间批量归一化

我们已经看到，对于训练深层的全连接网络来说批量归一化是非常有用的技术。如论文 (BatchNormalization.ipynb 中的链接) 中所建议的，批处理归一化也可以用于卷积网络，但是我们需要对其进行一些调整，该修改将称为 "空间批量归一化"。

通常，当我们对维数为 N 的最小批进行批归一化时接受的形状为 (N，D) 的输入，之后生成形状为 (N，D) 的输出。对于来自卷积层的数据，批归一化需要接受形状为 (N，C，H，W) 的输入，并产

生形状为（N，C，H，W）的输出，其中 N 维度为最小批大小而（H，W）维度是特征图的大小。

如果特征图是使用卷积生成的，那么我们期望每个特征通道的两个不同图像以及同一图像内不同位置之间的统计信息例如均值、方差相对一致。毕竟每个特征通道都是由相同的卷积滤波器产生的！因此，空间批量归一化通过计算最小批维度 N 以及空间维度 H 和 W 的统计信息，为每个 C 特征通道计算均值和方差。

[1] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015.

10.1 空间批量归一化：正向传播

在文件 daseCV/layers.py 中的 `spatial_batchnorm_forward` 函数里实现空间批归一化的正向传播。通过运行以下命令检查您的代码：

```python
np.random.seed(231)
# Check the training-time forward pass by checking means and variances
# of features both before and after spatial batch normalization

N, C, H, W = 2, 3, 4, 5
x = 4 * np.random.randn(N, C, H, W) + 10

print('Before spatial batch normalization:')
print('  Shape: ', x.shape)
print('  Means: ', x.mean(axis=(0, 2, 3)))
print('  Stds: ', x.std(axis=(0, 2, 3)))

# Means should be close to zero and stds close to one
gamma, beta = np.ones(C), np.zeros(C)
bn_param = {'mode': 'train'}
out, _ = spatial_batchnorm_forward(x, gamma, beta, bn_param)
print('After spatial batch normalization:')
print('  Shape: ', out.shape)
print('  Means: ', out.mean(axis=(0, 2, 3)))
print('  Stds: ', out.std(axis=(0, 2, 3)))

# Means should be close to beta and stds close to gamma
gamma, beta = np.asarray([3, 4, 5]), np.asarray([6, 7, 8])
```

```
out, _ = spatial_batchnorm_forward(x, gamma, beta, bn_param)
print('After spatial batch normalization (nontrivial gamma, beta):')
print('  Shape: ', out.shape)
print('  Means: ', out.mean(axis=(0, 2, 3)))
print('  Stds: ', out.std(axis=(0, 2, 3)))
```

```
Before spatial batch normalization:
  Shape:  (2, 3, 4, 5)
  Means:  [9.33463814 8.90909116 9.11056338]
  Stds:   [3.61447857 3.19347686 3.5168142 ]
After spatial batch normalization:
  Shape:  (2, 3, 4, 5)
  Means:  [ 6.18949336e-16   5.99520433e-16 -1.22124533e-16]
  Stds:   [0.99999962 0.99999951 0.9999996 ]
After spatial batch normalization (nontrivial gamma, beta):
  Shape:  (2, 3, 4, 5)
  Means:  [6. 7. 8.]
  Stds:   [2.99999885 3.99999804 4.99999798]
```

[34]:
```
np.random.seed(231)
# Check the test-time forward pass by running the training-time
# forward pass many times to warm up the running averages, and then
# checking the means and variances of activations after a test-time
# forward pass.
N, C, H, W = 10, 4, 11, 12

bn_param = {'mode': 'train'}
gamma = np.ones(C)
beta = np.zeros(C)
for t in range(50):
  x = 2.3 * np.random.randn(N, C, H, W) + 13
  spatial_batchnorm_forward(x, gamma, beta, bn_param)
bn_param['mode'] = 'test'
x = 2.3 * np.random.randn(N, C, H, W) + 13
a_norm, _ = spatial_batchnorm_forward(x, gamma, beta, bn_param)
```

```
# Means should be close to zero and stds close to one, but will be
# noisier than training-time forward passes.
print('After spatial batch normalization (test-time):')
print('  means: ', a_norm.mean(axis=(0, 2, 3)))
print('  stds: ', a_norm.std(axis=(0, 2, 3)))
```

```
After spatial batch normalization (test-time):
  means:  [-0.08034406  0.07562881  0.05716371  0.04378383]
  stds:  [0.96718744 1.0299714  1.02887624 1.00585577]
```

10.2 空间批量归一化：反向传播

在文件 daseCV/layers.py 中的函数 spatial_batchnorm_backward 里实现空间批量归一化的反向传播。运行以下命令以检查您的代码：

[35]:
```python
np.random.seed(231)
N, C, H, W = 2, 3, 4, 5
x = 5 * np.random.randn(N, C, H, W) + 12
gamma = np.random.randn(C)
beta = np.random.randn(C)
dout = np.random.randn(N, C, H, W)

bn_param = {'mode': 'train'}
fx = lambda x: spatial_batchnorm_forward(x, gamma, beta, bn_param)[0]
fg = lambda a: spatial_batchnorm_forward(x, gamma, beta, bn_param)[0]
fb = lambda b: spatial_batchnorm_forward(x, gamma, beta, bn_param)[0]

dx_num = eval_numerical_gradient_array(fx, x, dout)
da_num = eval_numerical_gradient_array(fg, gamma, dout)
db_num = eval_numerical_gradient_array(fb, beta, dout)

#You should expect errors of magnitudes between 1e-12~1e-06
_, cache = spatial_batchnorm_forward(x, gamma, beta, bn_param)
dx, dgamma, dbeta = spatial_batchnorm_backward(dout, cache)
print('dx error: ', rel_error(dx_num, dx))
print('dgamma error: ', rel_error(da_num, dgamma))
```

```
print('dbeta error: ', rel_error(db_num, dbeta))
```

dx error: 2.786648193872555e-07

dgamma error: 7.0974817113608705e-12

dbeta error: 3.275608725278405e-12

11 组归一化

在之前的 notebook 中，我们提到了"层归一化"是一种替代的归一化技术，它减轻了"批归一化"的批大小限制。但是，正如 [2] 的作者所观察到的，当与卷积层一起使用时，层归一化的性能不如批归一化：

> With fully connected layers, all the hidden units in a layer tend to make similar contributions to the final prediction, and re-centering and rescaling the summed inputs to a layer works well. However, the assumption of similar contributions is no longer true for convolutional neural networks. The large number of the hidden units whose receptive fields lie near the boundary of the image are rarely turned on and thus have very different statistics from the rest of the hidden units within the same layer.

[3] 的作者提出了一种中间技术。与"层归一化"相反，在"层归一化"中您对每个数据点的整个特征进行归一化，他们建议将每个数据点一致的特征划分为 G 组，然后对每个组的每个数据点进行归一化。

Visual comparison of the normalization techniques discussed so far (image edited from [3])

尽管在每一组中仍然存在贡献相等的假设，但作者假设这不是问题，因为在视觉识别的特征中出现了天生的分组。他们用来说明这一点的一个例子是，在传统的计算机视觉中，许多高性能的传统的特征都有明确分组在一起的术语。以 Histogram of Oriented Gradients[4] 为例——在计算每个空间局部块的直方图后，对每个块的直方图进行归一化处理，然后拼接在一起形成最终的特征向量。

现在，你将实现组归一化。请注意，你将在以下 cell 中实现的这种归一化技术是在 2018 年引入并发布到 ECCV 的，这是是一个正在进行且激动人心的研究领域!

[2] Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization." stat 1050 (2016): 21.

[3] Wu, Yuxin, and Kaiming He. "Group Normalization." arXiv preprint arXiv:1803.08494 (2018).

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In Computer Vision and Pattern Recognition (CVPR), 2005.

11.1   组归一化：正向传播

在文件 daseCV/layers.py 中的 spatial_groupnorm_forward 函数里实现组归一化的正向传播。通过运行以下命令检查您的代码：

```
[36]: np.random.seed(231)
      # Check the training-time forward pass by checking means and variances
      # of features both before and after spatial batch normalization

      N, C, H, W = 2, 6, 4, 5
      G = 2
      x = 4 * np.random.randn(N, C, H, W) + 10
      x_g = x.reshape((N*G,-1))
      print('Before spatial group normalization:')
      print('  Shape: ', x.shape)
      print('  Means: ', x_g.mean(axis=1))
      print('  Stds: ', x_g.std(axis=1))


      # Means should be close to zero and stds close to one
      gamma, beta = np.ones((1,C,1,1)), np.zeros((1,C,1,1))
      bn_param = {'mode': 'train'}


      out, _ = spatial_groupnorm_forward(x, gamma, beta, G, bn_param)
      out_g = out.reshape((N*G,-1))
      print('After spatial group normalization:')
      print('  Shape: ', out.shape)
      print('  Means: ', out_g.mean(axis=1))
      print('  Stds: ', out_g.std(axis=1))
```

```
Before spatial group normalization:
  Shape:  (2, 6, 4, 5)
  Means:  [9.72505327 8.51114185 8.9147544  9.43448077]
  Stds:  [3.67070958 3.09892597 4.27043622 3.97521327]
After spatial group normalization:
```

```
Shape:  (2, 6, 4, 5)
Means:  [-2.14643118e-16  5.25505565e-16  2.58126853e-16 -3.62672855e-16]
Stds:   [0.99999963 0.99999948 0.99999973 0.99999968]
```

11.2 空间组归一化：反向传播

在文件 daseCV/layers.py 中的 `spatial_groupnorm_backward` 函数里实现空间批量归一化的反向传播。运行以下命令以检查您的代码：

```
[37]: np.random.seed(231)
N, C, H, W = 2, 6, 4, 5
G = 2
x = 5 * np.random.randn(N, C, H, W) + 12
gamma = np.random.randn(1,C,1,1)
beta = np.random.randn(1,C,1,1)
dout = np.random.randn(N, C, H, W)


gn_param = {}
fx = lambda x: spatial_groupnorm_forward(x, gamma, beta, G, gn_param)[0]
fg = lambda a: spatial_groupnorm_forward(x, gamma, beta, G, gn_param)[0]
fb = lambda b: spatial_groupnorm_forward(x, gamma, beta, G, gn_param)[0]


dx_num = eval_numerical_gradient_array(fx, x, dout)
da_num = eval_numerical_gradient_array(fg, gamma, dout)
db_num = eval_numerical_gradient_array(fb, beta, dout)


_, cache = spatial_groupnorm_forward(x, gamma, beta, G, gn_param)
dx, dgamma, dbeta = spatial_groupnorm_backward(dout, cache)
#You should expect errors of magnitudes between 1e-12~1e-07
print('dx error: ', rel_error(dx_num, dx))
print('dgamma error: ', rel_error(da_num, dgamma))
print('dbeta error: ', rel_error(db_num, dbeta))
```

```
dx error:  6.345904562232329e-08
dgamma error:  1.0546047434202244e-11
dbeta error:  3.810857316122484e-12
```

12   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。

```python
import os

FOLDER_TO_SAVE = os.path.join('drive/My Drive/', FOLDERNAME)
FILES_TO_SAVE = ['daseCV/classifiers/cnn.py', 'daseCV/classifiers/fc_net.py',\
                 'daseCV/layers.py','daseCV/optim.py',\
                 'daseCV/solver.py','daseCV/layer_utils.py',\
                 'daseCV/data_utils.py','daseCV/fast_layers.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()))
```