Spatial batch normalization

In fully connected networks, we performed batch normalization on the activations. To do something equivalent on CNNs, we modify batch normalization slightly.

Normally batch-normalization accepts inputs of shape `(N, D)` and produces outputs of shape `(N, D)`, where we normalize across the minibatch dimension `N`. For data coming from convolutional layers, batch normalization accepts inputs of shape `(N, C, H, W)` and produces outputs of shape `(N, C, H, W)` where the `N` dimension gives the minibatch size and the `(H, W)` dimensions give the spatial size of the feature map.

How do we calculate the spatial averages? First, notice that for the `C` feature maps we have (i.e., the layer has `C` filters) that each of these ought to have its own batch norm statistics, since each feature map may be picking out very different features in the images. However, within a feature map, we may assume that across all inputs and across all locations in the feature map, there ought to be relatively similar first and second order statistics. Hence, one way to think of spatial batch-normalization is to reshape the `(N, C, H, W)` array as an `(N*H*W, C)` array and perform batch normalization on this array.

Since spatial batch norm and batch normalization are similar, it'd be good to at this point also copy and paste our prior implemented layers from HW #4. Please copy and paste your prior implemented code from HW #4 to start this assignment. If you did not correctly implement the layers in HW #4, you may collaborate with a classmate to use their implementations from HW #4. You may also visit TA or Prof OH to correct your implementation.

You'll want to copy and paste from HW #4: - layers.py for your FC network layers, as well as batchnorm and dropout. - layer_utils.py for your combined FC network layers. - optim.py for your optimizers.

Be sure to place these in the `nndl/` directory so they're imported correctly. Note, as announced in class, we will not be releasing our solutions.

If you use your prior implementations of the batchnorm, then your spatial batchnorm implementation may be very short. Our implementations of the forward and backward pass are each 6 lines of code.

In [1]:

```
## Import and setups

import time
import numpy as np
import matplotlib.pyplot as plt
from nndl.conv_layers import *
from utils.data_utils import get_CIFAR10_data
from utils.gradient_check import eval_numerical_gradient, eval_numerical_gradient_array
from utils.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))))
```

Spatial batch normalization forward pass

Implement the forward pass, `spatial_batchnorm_forward` in `nndl/conv_layers.py`. Test your implementation by running the cell below.

In [2]:

```
# 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:  [ 9.05477759 10.20247671 10.88945699]
  Stds:  [3.56072987 3.71327858 4.30309899]
After spatial batch normalization:
  Shape:  (2, 3, 4, 5)
  Means:  [ 1.11022302e-17  6.93889390e-17 -2.22044605e-17]
  Stds:  [0.99999965 0.99999967 0.9999997 ]
After spatial batch normalization (nontrivial gamma, beta):
  Shape:  (2, 3, 4, 5)
  Means:  [6. 7. 8.]
  Stds:  [2.99999895 3.99999866 4.99999851]
```

Spatial batch normalization backward pass

Implement the backward pass, `spatial_batchnorm_backward` in `nndl/conv_layers.py` . Test your implementation by running the cell below.

In [3]:

```
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)

_, 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:  7.549500110129972e-09
dgamma error:  1.0955086773552313e-11
dbeta error:  1.1595067532532086e-11
```

In [ ]:

```
def spatial_batchnorm_forward(x, gamma, beta, bn_param):
  """
  Computes the forward pass for spatial batch normalization.

  Inputs:
  - x: Input data of shape (N, C, H, W)
  - gamma: Scale parameter, of shape (C,)
  - beta: Shift parameter, of shape (C,)
  - bn_param: Dictionary with the following keys:
    - mode: 'train' or 'test'; required
    - eps: Constant for numeric stability
    - momentum: Constant for running mean / variance. momentum=0 means that
      old information is discarded completely at every time step, while
      momentum=1 means that new information is never incorporated. The
      default of momentum=0.9 should work well in most situations.
    - running_mean: Array of shape (D,) giving running mean of features
    - running_var Array of shape (D,) giving running variance of features

  Returns a tuple of:
  - out: Output data, of shape (N, C, H, W)
  - cache: Values needed for the backward pass
  """
  out, cache = None, None

  # ================================================================ #
  # YOUR CODE HERE:
  #   Implement the spatial batchnorm forward pass.
  #
  #   You may find it useful to use the batchnorm forward pass you
  #   implemented in HW #4.
  # ================================================================ #

  N, C, H, W = x.shape #get the shape of x

  #reshape the input to 2D array for Batch Normalization
  x = x.reshape(N, H, W, C)
  x = x.reshape(N * H * W, C)

  #do the batchnorm forward
  out, cache = batchnorm_forward(x, gamma, beta, bn_param)

  #reshape the output to (N, C, H, W)
  out = out.reshape((N, H, W, C)).transpose(0, 3, 1, 2)

  # ================================================================ #
  # END YOUR CODE HERE
  # ================================================================ #

  return out, cache


def spatial_batchnorm_backward(dout, cache):
  """
  Computes the backward pass for spatial batch normalization.

  Inputs:
  - dout: Upstream derivatives, of shape (N, C, H, W)
  - cache: Values from the forward pass

  Returns a tuple of:
  - dx: Gradient with respect to inputs, of shape (N, C, H, W)
  - dgamma: Gradient with respect to scale parameter, of shape (C,)
  - dbeta: Gradient with respect to shift parameter, of shape (C,)
  """
  dx, dgamma, dbeta = None, None, None

  # ================================================================ #
  # YOUR CODE HERE:
  #   Implement the spatial batchnorm backward pass.
  #
  #   You may find it useful to use the batchnorm forward pass you
  #   implemented in HW #4.
  # ================================================================ #

  N, C, H, W = dout.shape #get the shape of dout

  #reshape the dout
  dout = (dout.transpose(0, 2, 3, 1)).reshape(N * H * W, C)

  #do the batchnorm backward
  dx, dgamma, dbeta = batchnorm_backward(dout, cache)

  #reshape the output to de desired format
  dx = dx.reshape(N, C, H, W)

  # ================================================================ #
  # END YOUR CODE HERE
  # ================================================================ #
```

```
    return dx, dgamma, dbeta
```