In [ ]:
```
from .layers import *

"""
This code was originally written for CS 231n at Stanford University
(cs231n.stanford.edu).  It has been modified in various areas for use
in the
ECE 239AS class at UCLA.  This includes the descriptions of what code
to
implement as well as some slight potential changes in variable names t
o be
consistent with class nomenclature.  We thank Justin Johnson & Serena
Yeung for
permission to use this code.  To see the original version, please visi
t
cs231n.stanford.edu.
"""

def affine_relu_forward(x, w, b):
  """
  Convenience layer that performs an affine transform followed by a Re
LU

  Inputs:
  - x: Input to the affine layer
  - w, b: Weights for the affine layer

  Returns a tuple of:
  - out: Output from the ReLU
  - cache: Object to give to the backward pass
  """
  a, fc_cache = affine_forward(x, w, b)
  out, relu_cache = relu_forward(a)
  cache = (fc_cache, relu_cache)
  return out, cache


def affine_relu_backward(dout, cache):
  """
  Backward pass for the affine-relu convenience layer
  """
  fc_cache = cache[0]
  relu_cache = cache[1]
#  print("fc cache", fc_cache)
#  print(len(cache))

  da = relu_backward(dout, relu_cache)
  dx, dw, db = affine_backward(da, fc_cache)
  return dx, dw, db
```

```
def affine_batchnorm_relu_forward(x, w, b, gamma, beta, bn_params):
  """
  Performs affine transformation, batchnorm, and ReLU

  Returns all caches

  BN forward takes: def batchnorm_forward(x, gamma, beta, bn_param):

  """
  out, forward_cache = affine_forward(x, w, b)
#  print("beta received: ", beta.shape)
  out, batchnorm_cache = batchnorm_forward(out, gamma, beta, bn_params
)
#  print("got dim: ", out.dim)
  out, relu_cache = relu_forward(out)

  total_cache = (forward_cache,relu_cache, batchnorm_cache)
#  print("returning out dim: ", out.shape)
  return out, total_cache


def affine_batchnorm_relu_backward(dout, cache):
  """
  Backward pass
  def batchnorm_backward(dout, cache):
  def relu_backward(dout, cache):


  """
  #unpack the cache tuple
  forward_cache, relu_cache, batchnorm_cache  = cache

  dx = relu_backward(dout, relu_cache)
  dx, dgamma, dbeta = batchnorm_backward(dx, batchnorm_cache)
  dx, dw, db = affine_backward(dx, forward_cache)

  gradients = dx, dw, db, dgamma, dbeta
  return gradients
```