layer_utils.py

In [ ]:

```
from nndl.layers import *
from cs231n.fast_layers import *
from nndl.conv_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_forward_batchnorm(x, w, b, gamma, beta, bn_param):
  a, fc_cache = affine_forward(x, w, b)
  a, bn_cache = batchnorm_forward(a, gamma, beta, bn_param)
  out, relu_cache = relu_forward(a)
  cache = (fc_cache, bn_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_relu_backward_batchnorm(dout, cache):
  fc_cache, bn_cache, relu_cache = cache
  da = relu_backward(dout, relu_cache)
  dbn, dgamma, dbeta = batchnorm_backward(da, bn_cache)
  dx, dw, db = affine_backward(dbn, fc_cache)

  return dx, dw, db, dgamma, dbeta

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


"""
Functions for conv net without batchnorm
"""
def conv_relu_forward(x, w, b, conv_param):
#    conv_param = {'stride': 1, 'pad': (filter_size - 1) / 2}
  out, conv_cache = conv_forward_fast(x, w, b, conv_param)
  out, relu_cache = relu_forward(out)
  cache = (conv_cache, relu_cache)

  return out, cache

def conv_relu_backward(dout, cache):
  conv_cache, relu_cache = cache
  deriv = relu_backward(dout, relu_cache)
  dx, dw, db = conv_backward_fast(deriv, conv_cache)
  return dx, dw, db


#apply pooling
def conv_relu_pool_forward(x, w, b, conv_param, pool_param):
  out_conv_forward, conv_cache = conv_forward_fast(x, w, b, conv_param
)
  out_relu_forward, relu_cache = relu_forward(out_conv_forward)
  out, pool_cache = max_pool_forward_fast(out_relu_forward, pool_param
)

  cache = (conv_cache, relu_cache, pool_cache)
  return out, cache

def conv_relu_pool_backward(dout, cache):
  conv_cache, relu_cache, pool_cache = cache

  dpool = max_pool_backward_fast(dout, pool_cache)
  drelu = relu_backward(dpool, relu_cache)
  dx, dw, db = conv_backward_fast(drelu, conv_cache)

  return dx, dw, db

"""
Functions with batchnorm
"""
def conv_relu_forward_batchnorm(x, w, b, conv_param, gamma, beta, bn_p
aram):
```

```
  out, conv_cache = conv_forward_fast(x, w, b, conv_param)
  out, bn_cache = spatial_batchnorm_forward(out, gamma, beta, bn_param
)
  out, relu_cache = relu_forward(out)
  cache = (conv_cache, bn_cache, relu_cache)

  return out, cache

def conv_relu_backward_batchnorm(dout, cache):
  #relu back -> batchnorm back -> conv back
  conv_cache, bn_cache, relu_cache = cache

  deriv = relu_backward(dout, relu_cache)
  dbn, dgamma, dbeta = spatial_batchnorm_backward(deriv, bn_cache)
  dx, dw, db = conv_backward_fast(dbn, conv_cache)
  return dx, dw, db, dgamma, dbeta

def conv_relu_pool_forward_batchnorm(x, w, b, conv_param, pool_param,
gamma, beta, bn_param):
    #conv forward
    out_conv_forward, conv_cache = conv_forward_fast(x, w, b, conv_par
am)
    #batchnorm forward - def spatial_batchnorm_forward(x, gamma, beta,
bn_param):
    out_bn_forward, bn_cache = spatial_batchnorm_forward(out_conv_forw
ard, gamma, beta, bn_param)
    #relu forward
    out_relu_forward, relu_cache = relu_forward(out_bn_forward)
    #pool
#    print(pool_param, pool_param['pool_height'])
    out, pool_cache = max_pool_forward_fast(out_relu_forward, pool_par
am)

    cache = (conv_cache, bn_cache, relu_cache, pool_cache)
    return out, cache

def conv_relu_pool_backward_batchnorm(dout, cache):
  conv_cache, bn_cache, relu_cache, pool_cache = cache

  #pool -> relu -> batchnorm back -> conv
  dpool = max_pool_backward_fast(dout, pool_cache)
  drelu = relu_backward(dpool, relu_cache)

  dbn, dgamma, dbeta = spatial_batchnorm_backward(drelu, bn_cache)

  dx, dw, db = conv_backward_fast(dbn, conv_cache)

  grads = (dx, dw, db, dgamma, dbeta)
  return grads
```