1. Batch Normalization
   Derivation:

$$
\begin{aligned}
\hat{x}_{ij} &= \frac{x_{ij} - \mu_j}{(\sigma_j^2 + \epsilon)^2} \\
\mu_j &= \frac{1}{N}\sum_i x_{ij} \\
\sigma_j^2 &= \frac{1}{N}\sum_i (x_{ij} - \mu_j)^2 \\
y_{ij} &= \gamma_j \hat{x}_{ij} + \beta_j \\
\frac{\partial y_{ij}}{\partial x_{ij}} &= \frac{\partial y_{ij}}{\partial \hat{x}_{ij}}\frac{\partial \hat{x}_{ij}}{\partial x_{ij}} + (\sum_i \frac{\partial y_{ij}}{\partial \hat{x}_{ij}}\frac{\partial \hat{x}_{ij}}{\partial \mu_j})\frac{\partial \mu_j}{\partial x_{ij}} + (\sum_i \frac{\partial y_{ij}}{\partial \hat{x}_{ij}}\frac{\partial \hat{x}_{ij}}{\partial \sigma_j^2})\frac{\partial \sigma_j^2}{\partial x_{ij}} \\
\frac{\partial \sigma_j^2}{\partial x_{ij}} &= \frac{2}{N}(x_{ij} - \mu_j) - (\sum_i \frac{2}{N}(x_{ij} - \mu_j))\frac{1}{N} \\
&= \frac{2}{N}(x_{ij} - \mu_j) + \frac{2}{N^2}(\sum_i (x_{ij} - \mu_j)) \\
let\ a_j &= \frac{1}{(\sigma_j^2 + \epsilon)^{0.5}} \\
\frac{\partial y_{ij}}{\partial x_{ij}} &= \partial y_{ij}\gamma_j a_j + (\sum_i \partial y_{ij}\gamma_j(-a_j))(\frac{1}{D}) + (\sum_i \partial y_{ij}\gamma_j(\frac{-1}{2})\hat{x}_{ij}a_j^2)(\frac{\partial \sigma_j^2}{\partial x_{ij}}) \\
\frac{\partial y_{ij}}{\partial x_{ij}} &= \partial y_{ij}\gamma_j a_i - \frac{1}{N}(\sum_i \partial y_{ij}\gamma_j a_i) - \frac{a_j}{N}\hat{x}_{ij}(\sum_i \partial y_{ij}\gamma_j \hat{x}_{ij}) + \frac{a_j}{N^2}(\sum_i \partial y_{ij}\gamma_j \hat{x}_{ij})(\sum_i \hat{x}_{ij})
\end{aligned}
$$

   Notes:

   - can omit the last term with $\frac{1}{N^2}$ since it contributes little to the overall sum

Forward Pass:

```
mode = bn_param['mode']
eps = bn_param.get('eps', 1e-5)
momentum = bn_param.get('momentum', 0.9)

N, D = x.shape
running_mean = bn_param.get('running_mean', np.zeros(D, dtype=x.dtype))
running_var = bn_param.get('running_var', np.zeros(D, dtype=x.dtype))

out, cache = None, None
if mode == 'train':
    batch_mean = np.sum(x, axis=0) / N
    batch_var = np.sum(np.power(x-batch_mean, 2), axis=0) / N
    running_mean = (1-momentum) * batch_mean + (momentum) * running_mean
    running_var = (1-momentum) * batch_var + (momentum) * running_var
    x_hat = (x - batch_mean) /(np.sqrt(batch_var + eps))
    out = gamma * x_hat + beta

    cache = (x, x_hat, gamma, beta, batch_mean, batch_var, eps)
elif mode == 'test':
    x_hat = (x - running_mean)/(np.sqrt(running_var + eps))
    out = gamma * x_hat + beta
else:
    raise ValueError('Invalid forward batchnorm mode "%s"' % mode)

# Store the updated running means back into bn_param
bn_param['running_mean'] = running_mean
bn_param['running_var'] = running_var
```

Backward Pass:

```
N, D = dout.shape

x, x_hat, gamma, beta, batch_mean, batch_var, eps = cache

dbeta = np.sum(dout, axis=0)
dgamma = np.sum(dout * x_hat, axis=0)

a = 1.0/np.sqrt(batch_var + eps)

dx = dout * gamma * a
     - 1./N * a * gamma * (np.sum(dout, axis=0))
     - 1./N * a * gamma * x_hat * np.sum(dout * x_hat, axis=0)
     + 1./N**2 * a * gamma *
         np.sum(dout * x_hat, axis=0) * np.sum(x_hat, axis=0)
```

2. Layer Normalization
   Derivation:

$$
\begin{aligned}
\hat{x}_{ij} &= \frac{x_{ij} - \mu_i}{(\sigma_i^2 + \epsilon)^2} \\
\mu_i &= \frac{1}{D}\sum_j x_{ij} \\
\sigma_i^2 &= \frac{1}{D}\sum_j (x_{ij} - \mu_i)^2 \\
y_{ij} &= \gamma_j \hat{x}_{ij} + \beta_j \\
\frac{\partial y_{ij}}{\partial x_{ij}} &= \frac{\partial y_{ij}}{\partial \hat{x}_{ij}}\frac{\partial \hat{x}_{ij}}{\partial x_{ij}} + (\sum_j \frac{\partial y_{ij}}{\partial \hat{x}_{ij}}\frac{\partial \hat{x}_{ij}}{\partial \mu_i})\frac{\partial \mu_i}{\partial x_{ij}} + (\sum_j \frac{\partial y_{ij}}{\partial \hat{x}_{ij}}\frac{\partial \hat{x}_{ij}}{\partial \sigma_i^2})\frac{\partial \sigma_i^2}{\partial x_{ij}} \\
\frac{\partial \sigma_i^2}{\partial x_{ij}} &= \frac{2}{D}(x_{ij} - \mu_i) - (\sum_j \frac{2}{D}(x_{ij} - \mu_i))\frac{1}{D} \\
&= \frac{2}{D}(x_{ij} - \mu_i) + \frac{2}{D^2}(\sum_j (x_{ij} - \mu_i)) \\
let\ a_i &= \frac{1}{(\sigma_i^2 + \epsilon)^{0.5}} \\
\frac{\partial y_{ij}}{\partial x_{ij}} &= \partial y_{ij}\gamma_j a_i + (\sum_j \partial y_{ij}\gamma_j(-a_i))(\frac{1}{D}) + (\sum_j \partial y_{ij}\gamma_j(\frac{-1}{2})\hat{x}_{ij}a_i^2)(\frac{\partial \sigma_i^2}{\partial x_{ij}}) \\
\frac{\partial y_{ij}}{\partial x_{ij}} &= \partial y_{ij}\gamma_j a_i - \frac{1}{D}(\sum_j \partial y_{ij}\gamma_j a_i) - \frac{a_i}{D}\hat{x}_{ij}(\sum_j \partial y_{ij}\gamma_j \hat{x}_{ij}) + \frac{a_i}{D^2}(\sum_j \partial y_{ij}\gamma_j \hat{x}_{ij})(\sum_j \hat{x}_{ij})
\end{aligned}
$$

Notes:

- can omit the last term with $\frac{1}{D^2}$ since it contributes little to the overall sum

Forward Pass:

```
N, D = x.shape

sample_mean = np.sum(x, axis=1) / D
sample_var = np.sum(np.power(x-np.expand_dims(sample_mean, axis=1), 2),
                 axis=1) / D
x_hat = (x - np.expand_dims(sample_mean, axis=1)) /
            (np.sqrt(np.expand_dims(sample_var, axis=1) + eps))
out = gamma * x_hat + beta

cache = (x, x_hat, gamma, beta, sample_mean, sample_var, eps)
```

Backward Pass:

```
N, D = dout.shape

x, x_hat, gamma, beta, sample_mean, sample_var, eps = cache

dbeta = np.sum(dout, axis=0)
dgamma = np.sum(dout * x_hat, axis=0)

a = 1.0/np.sqrt(sample_var + eps) #dim: (N)

dx = dout * np.expand_dims(gamma, axis=0) * np.expand_dims(a, axis=1)
     - 1./D * np.sum(dout * np.expand_dims(a,axis=1) *
         np.expand_dims(gamma, axis=0), axis=1, keepdims=True)
     - 1./D * np.expand_dims(a, axis=1) * x_hat *
         np.sum(dout * np.expand_dims(gamma, axis=0) * x_hat,
                axis=1, keepdims=True)
     + 1./D**2 * np.expand_dims(a, axis=1) *
         np.sum(dout * np.expand_dims(gamma, axis=0) * x_hat,
                axis=1, keepdims=True)
         * np.sum(x_hat, axis=1, keepdims=True)
```

3. Group Normalization

Derivation:

$$\text{let } dim(x) = (N = samples, C = channels, H = height, W = width)$$

$$G_{size} = C/G, \text{ where C mod G =0}$$

$$\hat{x}_{ijkl} = \frac{x_{ij} - \mu_{ig}}{(\sigma_{ig}^2 + \epsilon)^2}, j \in [gG_{size}, (g+1)G_{size} - 1], g \in 0, .., G-1$$

$$\mu_{ig} = \frac{1}{HWG_{size}} \sum_{k=0,l=0,g=jG_{size}}^{k=H-1,l=W-1,g=(j+1)G_{size}-1} x_{igkl}, j \in 0, ..G-1$$

$$\sigma_{ig}^2 = \frac{1}{HWG_{size}} \sum_{k=0,l=0,g=jG_{size}}^{k=H-1,l=W-1,g=(j+1)G_{size}-1} (x_{igkl} - \mu_i g)^2$$

$$y_{ijkl} = \gamma_j \hat{x}_{ijkl} + \beta_j$$

$$\frac{\partial y_{ijkl}}{\partial x_{ijkl}} = \frac{\partial y_{ijkl}}{\partial \hat{x}_{ijkl}} \frac{\partial \hat{x}_{ijkl}}{\partial x_{ijkl}} + (\sum_{j=gG_{size},k=0,l=0}^{j=(g+1)G_{size}-1,k=H-1,l=W-1} \frac{\partial y_{ijkl}}{\partial \hat{x}_{ijkl}} \frac{\partial \hat{x}_{ijkl}}{\partial \mu_{ig}}) \frac{\partial \mu_{ig}}{\partial x_{ijkl}} + (\sum_{j=gG_{size},k=0,l=0}^{j=(g+1)G_{size}-1,k=H-1,l=W-1} \frac{\partial y_{ijkl}}{\partial \hat{x}_{ijkl}} \frac{\partial \hat{x}_{ijkl}}{\partial \sigma_{ig}^2}) \frac{\partial \sigma_{ig}^2}{\partial x_{ijkl}}, j \in [gG_{size}, (g+1)G_{size} - 1]$$

$$\frac{\partial \sigma_i^2}{\partial x_{ijkl}} = \frac{2}{HWG_{size}}(x_{ijkl} - \mu_{ig}) - (\sum_{j=gG_{size},k=0,l=0}^{j=(g+1)G_{size}-1,k=H-1,l=W-1} \frac{2}{HWG_{size}}(x_{ijkl} - \mu_{ig})) \frac{1}{HWG_{size}}$$

$$= \frac{2}{HWG_{size}}(x_{ijkl} - \mu_{ig}) + O((\frac{1}{HWG_{size}})^2)$$

$$let\ a_{ig} = \frac{1}{(\sigma_{ig}^2 + \epsilon)^{0.5}}$$

$$\frac{\partial y_{ijkl}}{\partial x_{ijkl}} = \partial y_{ijkl} \gamma_j a_{ig} + (\sum_{k=0,l=0,g=jG_{size}}^{k=H-1,l=W-1,g=(j+1)G_{size}-1} \partial y_{ijkl} \gamma_j (-a_{ig}))(\frac{1}{HWG_{size}}) + (\sum_{k=0,l=0,g=jG_{size}}^{k=H-1,l=W-1,g=(j+1)G_{size}-1} \partial y_{ijkl} \gamma_j (\frac{-1}{2}) \hat{x}_{ijkl} a_{ig}^2)(\frac{\partial \sigma_{ig}^2}{\partial x_{ijkl}})$$

$$\frac{\partial y_{ijkl}}{\partial x_{ijkl}} = \partial y_{ijkl} \gamma_j a_{ig} - \frac{1}{HWG_{size}}(\sum_{k=0,l=0,g=jG_{size}}^{k=H-1,l=W-1,g=(j+1)G_{size}-1} \partial y_{ijkl} \gamma_j a_{ig}) - \frac{a_{ig}}{HWG_{size}} \hat{x}_{ijkl} (\sum_{k=0,l=0,g=jG_{size}}^{k=H-1,l=W-1,g=(j+1)G_{size}-1} \partial y_{ijkl} \gamma_j \hat{x}_{ijkl}) + O((\frac{1}{HWG_{size}})^2)$$

Forward Pass:

```
- x: Input data of shape (N, C, H, W)
- gamma: Scale parameter, of shape (1,C,1,1)
- beta: Shift parameter, of shape (1,C,1,1)
- G: Integer mumber of groups to split into, should be a divisor of C
- gn_param: Dictionary with the following keys:
  - eps: Constant for numeric stability

eps = gn_param.get('eps',1e-5)

N,C,H,W = x.shape
out = np.zeros(x.shape)

g_size = C // G

x_reshape = np.reshape(x, (N,G,-1,H,W))

means = np.sum(x_reshape,
            axis=(2,3,4),
            keepdims=True) /
    (x_reshape.shape[2] * x_reshape.shape[3] * x_reshape.shape[4])

variances = np.sum(np.power(x_reshape-means, 2),
                axis=(2,3,4),
                keepdims=True) /
    (x_reshape.shape[2] * x_reshape.shape[3] * x_reshape.shape[4])
x_hat = np.reshape((x_reshape-means)/np.sqrt(variances + eps), (N,-1,H,W))

out = gamma * x_hat + beta

cache = (x, x_hat, gamma, beta, means, variances, eps, G)
```

Backward Pass:

```
x, x_hat, gamma, beta, means, variances, eps, G = cache

gamma_reshape = np.reshape(gamma, (1,G,-1,1))
beta_reshape = np.reshape(beta, (G,-1))

N,C,H,W = dout.shape
dx = np.zeros(dout.shape)

dgamma = np.zeros((C))
dbeta = np.zeros((C))
g_size = C // G

# partition into G groups of channels before computing
# derivatives exactly the same way as layer normalization,
# except on groups of channels instead of on all channels

dout_reshape = np.reshape(dout, (N,G,-1,H,W))

dbeta = np.sum(dout, axis=(0,2,3), keepdims=True)
dgamma = np.sum(dout * x_hat, axis=(0,2,3), keepdims=True)

partition_func = 1. / (g_size * H * W)

a = 1.0/np.sqrt(variances + eps) #dim: (N,G,1,1,1)
aa = np.reshape(np.broadcast_to(a,(N,G,g_size,H,W)), (G,C,H,W))

dx = dout * gamma * aa
     + np.reshape(
           np.broadcast_to(
               -1. * partition_func *
                   np.sum(np.reshape(dout * gamma *aa, (N,G,-1,H,W)),
                       axis=(2,3,4),
                       keepdims=True),
               (N,G,g_size,H,W)),
           (N,C,H,W))\
     -1. * partition_func * aa * x_hat *
     np.reshape(
         np.broadcast_to(
             np.sum(np.reshape(dout * gamma * x_hat, (N,G,-1,H,W)),
                 axis=(2,3,4),
                 keepdims=True),
             (N, G, g_size, H, W)),
         (N,C,H,W))
```

4. Dropout(Inverted)

Derivation:

$$mask = rand(dim = x.shape, p = prob_keep)$$
$$y_{ij..} = \frac{1}{p} x_{ij..} mask_{ij..}$$
$$\frac{\partial y_{ij..}}{\partial x_{ij..}} = \frac{1}{p} \partial y_{ij..} mask_{ij..}$$

Forward Pass:

```
p, mode = dropout_param['p'], dropout_param['mode']
if 'seed' in dropout_param:
    np.random.seed(dropout_param['seed'])

mask = None
out = None

if mode == 'train':
    mask = (np.random.rand(*x.shape) < p) / p
    out = mask * x

elif mode == 'test':
    out = x

cache = (dropout_param, mask)
out = out.astype(x.dtype, copy=False)
```

Backward Pass:

```
dropout_param, mask = cache
mode = dropout_param['mode']

dx = None

if mode == 'train':
    dx = dout * mask
elif mode == 'test':
    dx = dout
return dx
```

5. Weight Initialization

   Derivation:

$$
\begin{aligned}
Var(\sum_i w_i x_i) &= \sum_i^n Var(w_i, x_i) \\
&= \sum_i^n E[w_i]^2 Var(x_i) + E[x_i]^2 Var(w_i) + Var(w_i)Var(x_i) \\
&= \sum_i^n Var(w_i)Var(x_i) \text{ assuming 0 mean} \\
&= nVar(w_i)Var(x_i) \\
&= Var(\frac{1}{n^{0.5}} w_i)Var(x_i) \\
\hat{w}_i &= \frac{1}{n^{0.5}}
\end{aligned}
$$

6. Convolution

Derivation:
In context of ML with image inputs, ignoring padding and stride:
let g be the weights with $f \in$ filters, $c \in$ channels C with width $\hat{W}$, height $\hat{H}$
let x be the input with $c \in$ channels C with width W, height H

$$y_{i,f,a,b} = \sum_{a,b}^{W,H} \sum_{c,w=-\frac{\hat{W}}{2},h=-\frac{\hat{H}}{2}}^{C,\frac{\hat{W}}{2},\frac{\hat{H}}{2}} x_{i,c,a,b}\ g_{f,c,a-w,b-h}$$

Forward Pass:

```
N, C, H, W = x.shape
stride = conv_param['stride']
pad = conv_param['pad']
F, _, HH, WW = w.shape
H_prime = int(1 + (H + 2 * pad - HH) / stride)
W_prime = int(1 + (W + 2 * pad - WW) / stride)
out = np.zeros((N, F, H_prime, W_prime))

import itertools
# for i, f, hh, ww in itertools.product(
#     range(N), range(F), range(H_prime), range(W_prime)):
#         out[i, f, hh, ww] = np.sum(w[f, :]
#             * x_pad[i, :, stride*hh:stride*hh+HH, stride*ww:stride*ww+WW])
#             + b[f]
# partially vectorized for i
# for f, hh, ww in itertools.product(
#     range(F), range(H_prime), range(W_prime)):
#         out[:, f, hh, ww] = np.sum(w[f, :]
#             * x_pad[:, :, stride*hh:stride*hh+HH, stride*ww:stride*ww+WW],
#                 axis=(1,2,3))
#             + b[f]
# partially vectorized for f
# for i, hh, ww in itertools.product(
#     range(N), range(H_prime), range(W_prime)):
#         out[i, :, hh, ww] = np.sum(w[:, :]
#             * x_pad[i, :, stride*hh:stride*hh+HH, stride*ww:stride*ww+WW],
#                axis=(1,2,3))
#             + b[:]
# partially vectorized for i, f
for hh, ww in itertools.product(range(H_prime), range(W_prime)):
    #expand w to (1,F,C,WW,HH)
```

```
#expand x_pad to (N,1,C,W,H)
#contraction along axis: C,W(local),H(local)
#output axis: N,F,W_prime,H_prime
out[:, :, hh, ww] = np.sum( np.expand_dims(w,axis=0) *
    np.expand_dims(x_pad[:, :,
                   stride*hh:stride*hh+HH,
                   stride*ww:stride*ww+WW],
                   axis=1), axis=(2,3,4))
    + b[:]
```