https://github.com/brianwchh/back-propagation.git

```
void backward_batchnorm_layer(layer l, network net)
```

Back-pro at BN

$$\mu_k = \frac{1}{B*H*W} \sum_{b=0}^{B-1} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} C_k^l(b,i,j)$$

$$\hat{C}_{k,b,i,j}^l = \frac{C_{k,b,i,j}^l - \mu_k}{\sqrt{\sigma^2 + e}}$$

$$\sigma_k^2 = \frac{1}{B*H*W} \sum_{b=0}^{B-1} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} (C_k^l(b,i,j) - \mu_k)^2$$

$$y_{k,b,i,j} = \gamma_k * \hat{C}_{k,b,i,j}^l + \beta_k$$

Given: $\delta_{k,b,i,j}^l = \frac{\partial Loss}{\partial y_{k,b,i,j}^l}$

```
backward_bias(l.bias_updates, l.delta, l.batch, l.out_c, l.out_w*l.out_h);
```

$$\frac{\partial Loss}{\partial \beta_k^l}$$

$$\frac{\partial J}{\partial \beta_k} = \sum_b \sum_i \sum_j \frac{\partial J}{y_{k,b,i,j}} * \frac{\partial y_{k,b,i,j}}{\beta_k}$$

$$= \sum_b \sum_i \sum_j \frac{\partial J}{y_{k,b,i,j}} * 1$$

下面這段代碼即實現上面的公式。把當前 output channel index n 的所有 delta 值相加，即爲此 output channel 的偏置值的 update。

```
{
    int i,b;
    for(b = 0; b < batch; ++b){
        for(i = 0; i < n; ++i){ // n is the output channel index
            bias_updates[i] += sum_array(delta+size*(i+b*n), size);
        }
    }
}
```

```
backward_scale_cpu(l.x_norm, l.delta, l.batch, l.out_c, l.out_w*l.out_h, l.scale_updates);
```

$$\frac{\partial Loss}{\partial \gamma_k^l}$$

$$\frac{\partial J}{\partial \gamma_k} = \sum_b \sum_i \sum_j \frac{\partial J}{y_{k,b,i,j}} * \frac{\partial y_{k,b,i,j}}{\gamma_k}$$

$$= \sum_b \sum_i \sum_j \frac{\partial J}{y_{k,b,i,j}} * \hat{C}^l_{k,b,i,j}$$

```c
void backward_scale_cpu(float *x_norm, float *delta, int batch, int n, int size, float *scale_updates)
{
    int i,b,f;
    for(f = 0; f < n; ++f){   // f is the input channel index
        float sum = 0;
        for(b = 0; b < batch; ++b){
            for(i = 0; i < size; ++i){ // size = w*h
                int index = i + size*(f + n*b);
                sum += delta[index] * x_norm[index];
            }
        }
        scale_updates[f] += sum;
    }
}
```

對 batch-norm 輸入的求導數，用於計算前一級的 delta map . 詳細推導過程見 techshare 的 pdf 文件。

$$\frac{\partial Loss}{\partial C^l_{b,i,j}} = \delta^l(b,i,j) * \gamma * \frac{1}{\sqrt{\sigma^2 + e}}$$

$$+ \frac{1}{B*N*M} * \sum_{b'=0}^{B-1} \sum_{i'=0}^{H-1} \sum_{j'=0}^{W-1} \gamma * \delta^l(b',i',j') * \frac{-1}{\sqrt{\sigma^2 + e}}$$

$$+ \frac{1}{B*N*M} * 2 * (C^l_{b,i,j} - \mu_{ch}) * \frac{0.5}{(\sigma^2 + e)^{\frac{-3}{2}}} * \sum_{b'=0}^{B-1} \sum_{i'=0}^{H-1} \sum_{j'=0}^{W-1} \gamma * \delta^l(b',i',j') * (C^l_{b',i',j'} - \mu_k)$$

```c
// scale_bias(l.delta, l.scales, l.batch, l.out_c, l.out_h*l.out_w);
void scale_bias(float *output, float *scales, int batch, int n, int size)
{
    int i,j,b;
    for(b = 0; b < batch; ++b){
        for(i = 0; i < n; ++i){  // input channel index
            for(j = 0; j < size; ++j){
                output[(b*n + i)*size + j] *= scales[i];
            }
        }
    }
}
```

這部分代碼計算 每個 delta(b,i,j)*gamma $\delta_k^l(b,i,j)*\gamma_k$ 上面公式省略了下標 k，k 爲 channel 的 index。

執行完此代碼，l.delta_k(b,i,j)包含的值爲 $\delta_k^l(b,i,j)*\gamma_k$

```c
// mean_delta_cpu(l.delta, l.variance, l.batch, l.out_c, l.out_w*l.out_h, l.mean_delta);
void mean_delta_cpu(float *delta, float *variance, int batch, int filters, int spatial, float *mean_delta)
{

    int i,j,k;
    for(i = 0; i < filters; ++i){
        mean_delta[i] = 0;
        for (j = 0; j < batch; ++j) {
            for (k = 0; k < spatial; ++k) {
                int index = j*filters*spatial + i*spatial + k;
                mean_delta[i] += delta[index];
            }
        }
        mean_delta[i] *= (-1./sqrt(variance[i] + .00001f));
    }
}
```

此代碼實現的是 mean_delta

$$\sum_{b'=0}^{B-1}\sum_{i'=0}^{H-1}\sum_{j'=0}^{W-1}\gamma*\delta^l(b',i',j')*\frac{-1}{\sqrt{\sigma^2+e}}$$

注意 mean_delta 的維度，1×#_of_filters

```c
// variance_delta_cpu(l.x, l.delta, l.mean, l.variance, l.batch, l.out_c, l.out_w*l.out_h, l.variance_delta);
void  variance_delta_cpu(float *x, float *delta, float *mean, float *variance, int batch, int filters, int spatial, float *variance_delta)
{

    int i,j,k;
    for(i = 0; i < filters; ++i){
        variance_delta[i] = 0;
        for(j = 0; j < batch; ++j){
            for(k = 0; k < spatial; ++k){
                int index = j*filters*spatial + i*spatial + k;
                variance_delta[i] += delta[index]*(x[index] - mean[i]);
            }
        }
        variance_delta[i] *= -.5 * pow(variance[i] + .00001f, (float)(-3./2.));
    }
}
```

此代碼實現的是：

注意 variance_delta 的維度，1×#_of_filters

```c
// normalize_delta_cpu(l.x, l.mean, l.variance, l.mean_delta, l.variance_delta, l.batch, l.out_c, l.out_w*l.out_h, l.delta);
void normalize_delta_cpu(float *x, float *mean, float *variance, float *mean_delta, float *variance_delta, int batch, int filters, int spatial, float *delta)
{
    int f, j, k;
    for(j = 0; j < batch; ++j){
        for(f = 0; f < filters; ++f){
            for(k = 0; k < spatial; ++k){
                int index = j*filters*spatial + f*spatial + k;
                delta[index] = delta[index] * 1./(sqrt(variance[f] + .00001f)) \
                + variance_delta[f] * 2. * (x[index] - mean[f]) / (spatial * batch) + mean_delta[f]/(spatial*batch);
            }
        }
    }
}
```

此代碼實現的是整個 $\dfrac{\partial Loss}{\partial C^l_{b,i,j}}$

```c
delta[index] * 1./(sqrt(variance[f] + .00001f))
```

這項即: $= \delta^l(b,i,j)*\gamma*\dfrac{1}{\sqrt{\sigma^2+e}}$

```c
+ variance_delta[f] * 2. * (x[index] - mean[f]) / (spatial * batch)
```

這項即:

$$+\frac{1}{B*N*M}*2*(C^l_{b,i,j}-\mu_{ch})*\frac{0.5}{(\sigma^2+e)^{\frac{-3}{2}}}*\sum_{b'=0}^{B-1}\sum_{i'=0}^{H-1}\sum_{j'=0}^{W-1}\gamma*\delta^l(b',i',j')*(C^l_{b',i',j'}-\mu_k)$$

```c
mean_delta[f]/(spatial*batch)
```

這項即: $+\dfrac{1}{B*N*M}*\displaystyle\sum_{b'=0}^{B-1}\sum_{i'=0}^{H-1}\sum_{j'=0}^{W-1}\gamma*\delta^l(b',i',j')*\dfrac{-1}{\sqrt{\sigma^2+e}}$

至此 $\dfrac{\partial Loss}{\partial C^l_{b,i,j}}$ 準備就緒,