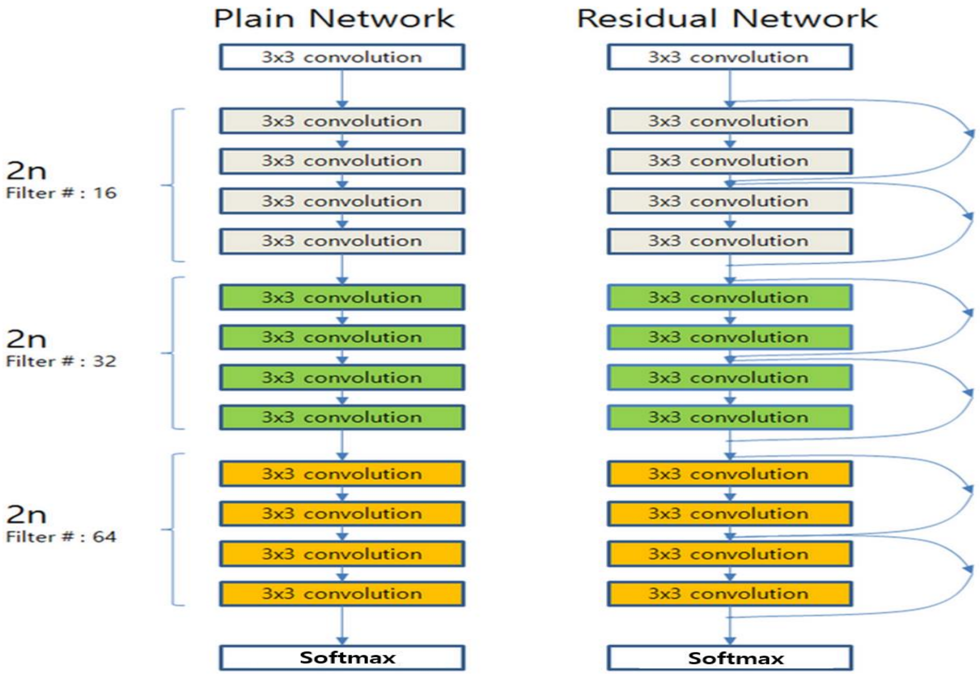
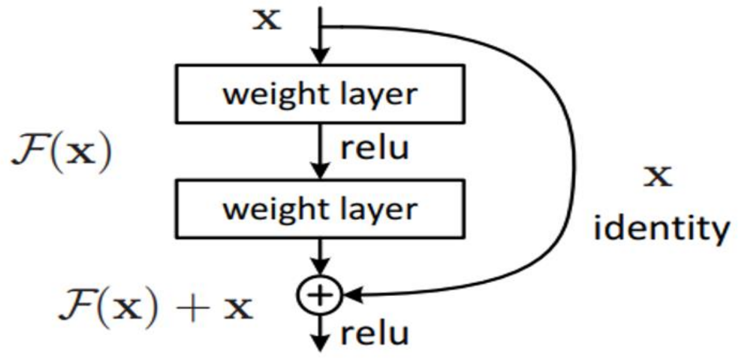
Resnet – Residual Blcok 쉽게 이해하기 by pytorch

1. Plain Network에서 발생하는 문제점 : 단순히 Layer를 깊게 쌓아서 발생하는 문제
   - Vanishing Gradient (기울기소실), Overfitting(과적합)
     -> Relu, Batch Normalization 등 기법 사용

2. 논문 : Deep Residual Learning for Image Recognition by K He

- Resnet은 Block 단위로 Param를 전달하기 전에 이전의 값을 더하는 방식
- Weight layer를 통과한 F(x)와 weight layer를 통과하지 않는 x의 합을 Residual mapping 이라고 하고, 그림의 구조를 Residual Block라고 하며, 이 Block이 쌓이면 Residual Network(Resnet)라고 한다.





Residual Block

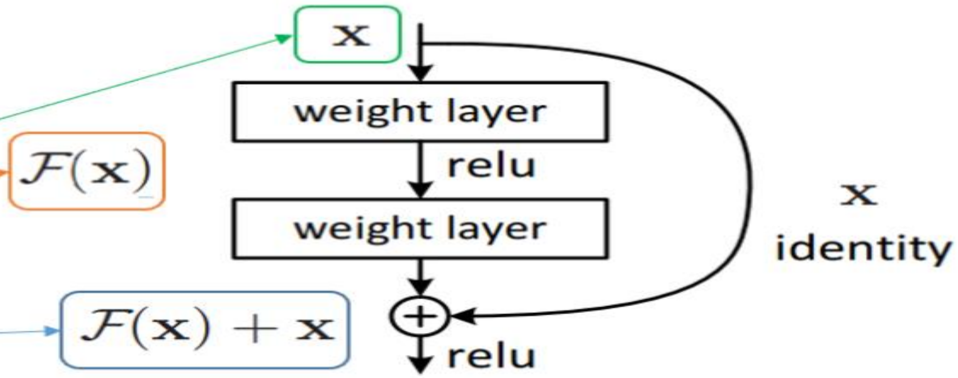
$$\textbf{Residual } \boldsymbol{Mapping} = \boldsymbol{f(x) + x}$$

1. Class Residual_Block(nn.Module)과 BottleNeck의 원리

Residual Block

```
class Residual_Block(nn.Module):
  def __init__(self, in_dim, mid_dim, out_dim):
    super(Residual_Block,self).__init__()
    # Residual Block
    self.residual_block = nn.Sequential(
        nn.Conv2d(in_dim, mid_dim, kernel_size=3, padding=1),
        nn.ReLU,
        nn.Conv2d(mid_dim, out_dim, kernel_size=3, padding=1),
    )
    self.relu = nn.ReLU()

  def forward(self, x):
   out = self. residual_block(x)  # F(x)
    out = out + x  # F(x) + x
    out = self.relu(out)
    return out
```



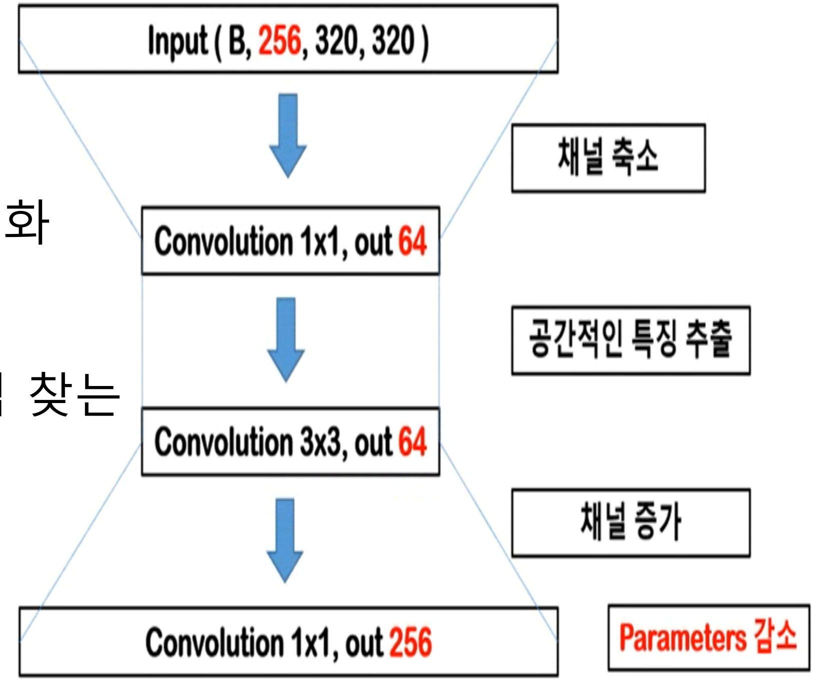
2. Convolution의 Parameter 계산

- Convolution Parameters = Kernel Size x Kernel Size x Input Channel x Output Channel
- BottleNeck의 핵심 : 1x1 convolution이다.
  -> 1x1 convolution의 paramete는 1x1x (input channel) x (output channel)
  -> 1x1 convolution은 연산량이 작기 때문에 Feature map(output channel)을 줄이거나 키울때 사용된다.

3. BottleNeck의 구조

1) Input channel = 256인, 320x320 Input Image가 있다고 가정한다.
   → input 256을 output channel 64로 차원, 채널 축소한다.
2) Channel Compression(채널압축)
   → Input channel 256을 Output Channel 64로 채널을 강제로 축소(연산량을 줄이기 위함)
   → 1x1 convolution에는 spatial(공간적인) 특징이 없다.
   → spatial 공간적인 특징 추출을 위해서 kernel이 최소 2이상이 되어야 한다.

3) 특징 추출
   → 3x3 convolution은 특성을 추출하는 역할을 하는데,
   → 3x3 convolution 연산 = 3 x 3 x input channel x output channel 이다.
   → 위 연산은 1x1 보다 9배 연산량이 많기 때문에, 1x1 convolution에서 채널을 줄인 후에 3x3 에서 특성을 추출한다.
4) 채널증가 : input channel 64에서 output channel 256로 증가한다.
   → CNN은 Feature map의 특성이 많을수록 학습이 잘되기 때문에, 1x1 convolution으로 강제적으로 채널을 증가시켜 준다.
   → BottleNeck의 구조는 1x1 convolution으로 장난치면서 연산량 최소화
   → 하지만 강제로 채널을 줄이고, 늘리는 것은 정보손실을 일으킨다.
   : 즉, 정보손실 = 모델의 정확성 감소
   → 연상량과 정보손실은 서로 tradeoff 관계이기 때문에 서로의 합의점 찾는 것이 중요하다.



3. BottleNeck의 구조

5) Parameter 확인

- Standard : 채널수가 64로 3x3 convolution을 2번 통과했다.
  * 참고) parmeters 계산하는 방법도 잘 나와있음(googling)
- BottleNeck : 채널수가 256으로 1x1, 3x3, 1x1,순으로 convolution을 통과했다.
  → Channel수는 Standard에 비해 4배 많지만, parameter 수를 보면 세배정도 작다.

cf) Pytorch Code

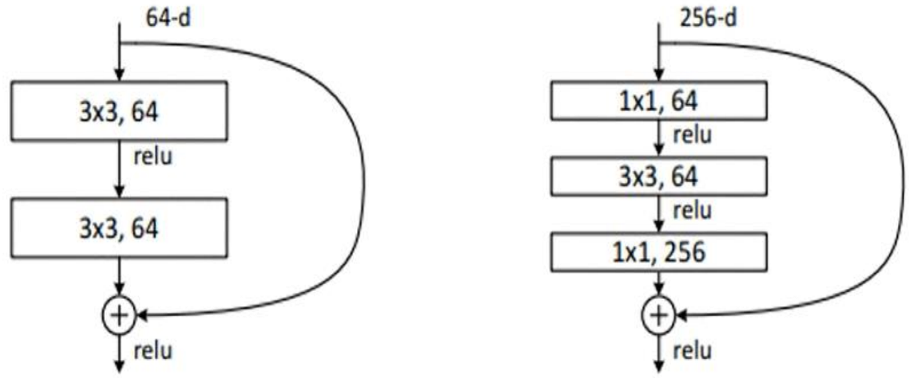
```python
# standard
class Standard(nn.Module):
    def __init__(self, in_dim=256, mid_dim=64, out_dim=64):
        super(BuildingBlock, self).__init__()
        self.building_block = nn.Sequential(
            nn.Conv2d(in_channels=in_dim, out_channels=mid_dim, kernel_size=3, padding=1, bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels=mid_dim, out_channels=out_dim, kernel_size=3, padding=1, bias=False),
        )
        self.relu = nn.ReLU()

    def forward(self, x):
        fx = self.building_block(x)  # F(x)
        out = fx + x  # F(x) + x
        out = self.relu(out)
        return out
```

```python
# BottleNeck
class BottleNeck(nn.Module):
    def __init__(self, in_dim=256, mid_dim=64, out_dim=256):
        super(BottleNeck, self).__init__()
        self.bottleneck = nn.Sequential(
            nn.Conv2d(in_channels=in_dim, out_channels=mid_dim, kernel_size=1, bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels=mid_dim, out_channels=mid_dim, kernel_size=3, padding=1, bias=False),
            nn.ReLU(),
            nn.Conv2d(in_channels=mid_dim, out_channels=in_dim, kernel_size=1, bias=False),
        )

        self.relu = nn.ReLU()

    def forward(self, x):
        fx = self.bottleneck(x)  # F(x)
        out = fx + x  # F(x) + x
        out = self.relu(out)
        return out
```



Standard **BottleNeck**

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 320, 320]	147,456
ReLU-2	[-1, 64, 320, 320]	0
Conv2d-3	[-1, 64, 320, 320]	36,864
Conv2d-4	[-1, 64, 320, 320]	147,456
ReLU-5	[-1, 64, 320, 320]	0

Total params: 331,776
Trainable params: 331,776
Non-trainable params: 0

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 320, 320]	16,384
ReLU-2	[-1, 64, 320, 320]	0
Conv2d-3	[-1, 64, 320, 320]	36,864
ReLU-4	[-1, 64, 320, 320]	0
Conv2d-5	[-1, 256, 320, 320]	16,384
Conv2d-6	[-1, 256, 320, 320]	65,792
ReLU-7	[-1, 256, 320, 320]	0

Total params: 135,424
Trainable params: 135,424
Non-trainable params: 0

4. Batch Normalization

1) Gradient Vanishing or Exploding 문제

- 신경망에서 학습 시 Gradient 기반의 방법들은 파라미터 값의 작은 변화가 신경망 출력에 얼마나 영향을 미칠 것인가를 기반으로 파라미터 값을 학습시키게 된다.
- 파라미터 값의 변화가 신경망 결과의 매우 작은 변화를 미치게 될 경우 파라미터를 효과적으로 학습시킬 수 없게 된다.
- Gradient = 미분값, 즉 변화량을 의미하는데 매우 작아지거나(vanishing) 커진다면(Exploding) 신경망을 효과적으로 학습시키지 못하고, Error rate가 낮아지지 않고 수렴해버리는 문제가 발생

2) 활성화 함수 : sigmoid, tanh, relu 등

- 비선형적인 방식으로 입력값을 매우 작은 출력값의 범위로 squash 한다.
- 가령 sigmoid 함수는 실수 범위의 수를 [0,1]로 mapping 해버린다. 매우 넓은 입력값의 범위가 극도로 작은 범위값으로 mapping 된다.
- 이를 위해 자주 쓰이는 것이 RELU(Recitified Linear Unit)이다.

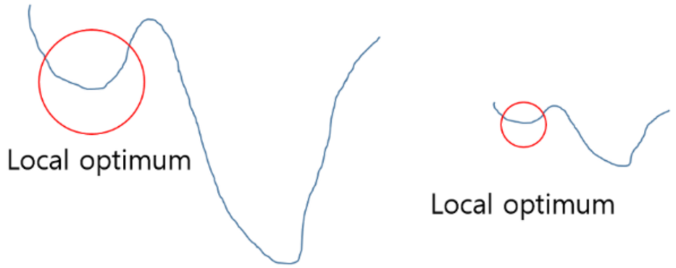
3) 배치 정규화 Batch Normalization

- 위의 방법보다는 '학습하는 과정 자체를 전체적으로 안정화하여 학습속도를 가속화 시킬 수 있는 근본적인 방법이다.
- 정규화를 하는 목적 : 학습을 더 빨리 하기 위해서 or Local optimum 문제에 빠지는 가능성을 줄이기 위해

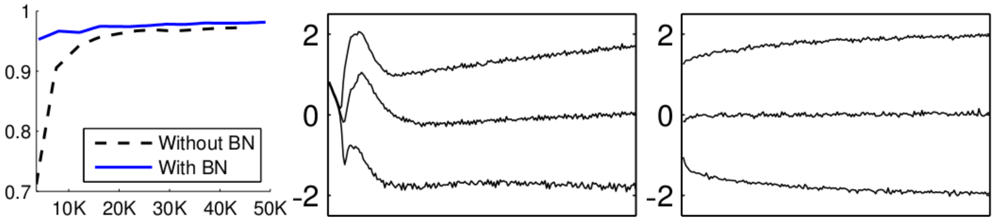
  → global optimum 지점을 찾지 못하고 local optimum에 머물러 있게 되는 문제가 발생하게 되는데, 이를 방지하기 위해!!

4) Internal Covariance Shift : 학습에서 불안정화가 일어나는 이유이다.

- 네트워크의 각 레이어나 Activation 마다 입력값의 분산이 달라지는 현상을 뜻한다.
- Covariance Shift : 이전 레이어의 파라미터 변화로 인하여 현재 레이어 입력의 분포가 바뀌는 현상
- Internal Covariance shift : 레이어를 통과할 때마다 Covariance shift가 일어나면서 입력의 분포가 약간식 변하는 현상



(좌) Normalization 적용 전 / (우) Normalization 적용 후

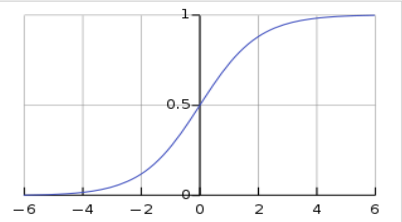


(a) (b) Without BN (c) With BN

4. Batch Normalization

5) Whitening 의 문제점

- 이 현상을 막기 위해 각 레이어 입력의 분산을 평균0, 표준편차 1을 입력값으로 정규화 시키는 방법을 생각한다.
- 들어오는 입력값의 특징을 uncorrelated하게 만들어주고, 분산을 1로 만들어준다.
- 단순하게 normalize 할 경우 문제가 생긴다. Activation function의 비선형성(우측 그래프 참고)이 없어질 수 도 있다. (sigmoid 함수 예)
- N(0,1) 이므로 95% 의 압력은 sigmoid 그래프의 중간(x=(-1.96, 1.96)구간)에 속하고 이 부분이 선형이다.
- Z = WX + b 라고 했을 때, normalize 하기 위해 Z-E(Z)를 하게 되면 parameter b 의 영향은 완전히 무시된다. 이를 위해 b를 대신하는 parameter β 를 추가한다. β가 normalization을 통해 상쇄되는 b의 역할을 대신하고, r가 scaling factor로 적용된다
- mini-batch의 평균과, 분산을 이용해서 normaliza 후 , scale and shift를 r , β를 통해 실행한다.
- 이렇게 normalize 된 값을 활성화 함수에 입력으로 사용하고 최종 출력물을 다음 층에 입력으로 사용한다.



6) 배치 정규화

- whitening 문제점을 해결하도록 한 트릭이 배치 정규화다.
- 평균과 분산을 조정하는 과정이 별도의 과정으로 떼어진 것이 아니라, 신경망 안에 포함되어 학습 시 평균과 분산을 조정하는 과정 역시 같이 조절된다는 점이 whitening과는 구별된다.
- 즉, 각 레이어마다 정규화 하는 레이어를 두어, 변형된 분포가 나오지 않도록 조절하게 하는 것이 배치 정규화이다.

- 간단히 말하자면 미니배치의 평균과 분산을 이용해서 정규화 한 뒤 , scale 및 shift를 감마값, 베타값을 통해 실행한다.
  → 감마값, 베타값 : 학습가능한 변수이고, backpropagation을 통해서 학습이 된다.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

배치 정규화의 입력 및 출력 값

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_\mathcal{B}) \cdot \frac{-1}{2}(\sigma_\mathcal{B}^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_\mathcal{B}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}\right) + \frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_\mathcal{B})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{2(x_i - \mu_\mathcal{B})}{m} + \frac{\partial \ell}{\partial \mu_\mathcal{B}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

감마, 베타에 대한 Backpropatation 수식

4. Batch Normalization

7) BN된 값

- 정규화된 값을 활성화 함수의 입력으로 사용하고, 최종 출력 값을 다음 레이어의 입력으로 사용한다.
- 기존 output = g(Z), Z = WX + b 식은 output = g(BN(Z)), Z = WX + b로 변경된다.
- 입실론(ε)은 계산할 때 분모가 0이 되는 것을 막기위한 수치적 안정성을 보장하기 위한 아주 작은 숫자이다.
- 감마(r)값 : scale에 대한 값이며, 베타(β)값 : shift transform 에 대한 값이다.
  → 데이터를 계속 정규화할 시 활성화 함수의 비선형 같은 성질을 잃게 되는데 이러한 문제를 완화하기 위함

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

배치 정규화의 입력 및 출력 값

8) Inference 시의 배치 정규화 : Train & Test

- 학습(train)시에는 배치정규화의 미니배치의 평균과 분산을 이용할 수 있지만, 추론(inference) 및 테스트 할 때에는 이용할 수 없다.
- inference 시 입력되는 데이터의 평균과 분산을 이용하면 배치정규화가 온전하게 이루어지지 않는다.
  → 배치정규화를 통해 수행하고자 한 것이.... 학습되는 동안 모델이 추정한 입력데이터 분포의 평균과 분산으로 정규화를 하고자 한것인데, inference 시에 입력되는 값을 통해서 정규화를 하게 되면 모델이 학습을 통해서 입력 데이터의 분포를 추정하는 의미 자체가 없어지게 된다.
  → 즉, inference 에서는 결과를 deterministic 하게 하기 위해서 고정된 평균과, 분산을 이용하여 정규화를 수행한다.
  그래서 train과 test 모드를 따로 두는 이유이기도 한다.

4. Batch Normalization

8) Inference 시의 배치 정규화 : Train & Test (계속)

- 그래서 이러한 문제를 미리 저장해둔 미니 배치의 이동평균(moving average)를 사용하여 해결한다.
- 즉, inference 전에 학습 시에 미리 미니배치를 뽑을 때, sample mean, sample variance를 이용하여 각각의 이동평균을 구해야한다. 위 수식에서 inference 시에 평균은 각 미니 배치에서 구한 평균들의 평균을 사용하고, 분산은 분산의 평균에 m/(m-1)을 곱해주게 된다.
- 이를 곱하는 이유는 통계학적으로 unbiased variance에는 'Bessel's correction'을 통해 보정을 해주는 것이다. 이는 학습 전체 데이터에 대한 분산이 아니라 미니 배치들의 분산으로 전체 분산 추정할 때 통계학적으로 보정을 위해 Bessel의 보정값을 곱해주는 방식으로 추정

9) CNN 구조에서의 배치 정규화

- CNN에서 활성화 함수가 입력되기 전에 WX+b 로 가중치가 적용될 시, b의 역할을 베타가 대신할 수 있기에 b를 삭제한다.
- CNN의 경우 성질을 유지시키고 싶기 때문에 각 채널을 기준으로 각각의 감마와 베타를 만들게 된다.

예) 미니배치가 m채널 , 사이즈가 n인 Convolution layer 에서 배치정규화를 적용시 적용 후 특징 맵의 사이즈가 pxq인 경우, 각 채널에 대해 mxpxq개의 스칼라값에 대해 평균과 분산을 구한다.(즉, nxmxpxq개의 스칼라 값). 최종적으로 감마,베타 값은 각 채널에 대해 한 개씩, 총 n개의 독립적인 배치 정규화 변수 쌍이 생기게 된다.

→ 즉, convolution kernel 하나는 같은 파라미터 감마, 베타를 공유하게 된다.

4. Batch Normalization

10) 결론 : 알고리즘(우측), 배치정규화 장점

- 신경망 레이어의 중간 중간에 위치하게 되어 **학습을 통해 감마, 베타를 구할 수 있음**
- 단순하게 평균과 분산을 구하는 것이 아니라 **감마(Scale), 베타(Shift) 를 통한 변환을 통해 비선형 성질을 유지 하면서 학습함**
- **Internal Covariate Shift 문제**로 인해 신경망이 깊어질 경우 학습이 어려웠던 문제점을 **해결**
- gradient 의 스케일이나 초기 값에 대한 dependency 가 줄어들어 Large Learning Rate 를 설정할 수 있기 떄문에 결과적으로 **빠른 학습 가능함**.
- **기존 방법에서 learning rate 를 높게 잡을 경우 gradient 가 vanish/explode 하거나 local minima 에 빠지는 경향이 있었는데 이는 scale 때문이었으며, 배치 정규화 사용시 propagation 시 파라미터의 scale 에 영향을 받지 않게 되기 때문에 learning rate 를 높게 설정할 수 있는 것**
- regularization 효과가 있기 때문에 **dropout 등의 기법을 사용하지 않아도 됨** (효과가 같음)
- 학습 시 Deterministic 하지 않은 결과생성 및 Learning Rate Decay 를 더 느리게 설정 가능
- 입력의 범위가 고정되어 saturating 한 활성화 함수를 써도 saturation 문제가 일어나지 않음.

  → 여기서 saturation 문제란 가중치의 업데이트가 없어지는 현상임

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

$N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$ // Training BN network
for $k = 1 \dots K$ **do**
  Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
  Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
end for
Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
$N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$ // Inference BN network with frozen // parameters
for $k = 1 \dots K$ **do**
  // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
  Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

  In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
end for

Algorithm 2: Training a Batch-Normalized Network

5. 활성화 함수 : Relu

10) 결론 : 알고리즘(우측), 배치정규화 장점

- 신경망 레이어의 중간 중간에 위치하게 되어 **학습을 통해 감마, 베타를 구할 수 있음**
- 단순하게 평균과 분산을 구하는 것이 아니라 **감마(Scale), 베타(Shift) 를 통한 변환을 통해 비선형 성질을 유지 하면서 학습함**
- **Internal Covariate Shift 문제**로 인해 신경망이 깊어질 경우 학습이 어려웠던 문제점을 **해결**
- gradient 의 스케일이나 초기 값에 대한 dependency 가 줄어들어 Large Learning Rate 를 설정할 수 있기 때문에 결과적으로 **빠른 학습 가능함**.
- **기존 방법에서 learning rate 를 높게 잡을 경우 gradient 가 vanish/explode 하거나 local minima 에 빠지는 경향이 있었는데 이는 scale 때문이었으며, 배치 정규화 사용시 propagation 시 파라미터의 scale 에 영향을 받지 않게 되기 때문에 learning rate 를 높게 설정할 수 있는 것**
- regularization 효과가 있기 때문에 **dropout 등의 기법을 사용하지 않아도 됨** (효과가 같음)
- 학습 시 Deterministic 하지 않은 결과생성 및 Learning Rate Decay 를 더 느리게 설정 가능
- 입력의 범위가 고정되어 saturating 한 활성화 함수를 써도 saturation 문제가 일어나지 않음.

  → 여기서 saturation 문제란 가중치의 업데이트가 없어지는 현상임