

A spiral binding on the left side of a notebook page, consisting of a series of blue and white rings.

CNN 세션 제 2강

주제: 학습을 좀 더 잘 하고싶어

목차

1. 오버피팅과 언더피팅 +여러 개념

2. 오버피팅 해결방법

2-1: Regularization

2-2: Dropout

3. Data Augmentation

4. Initializtion

5. Input Normalization

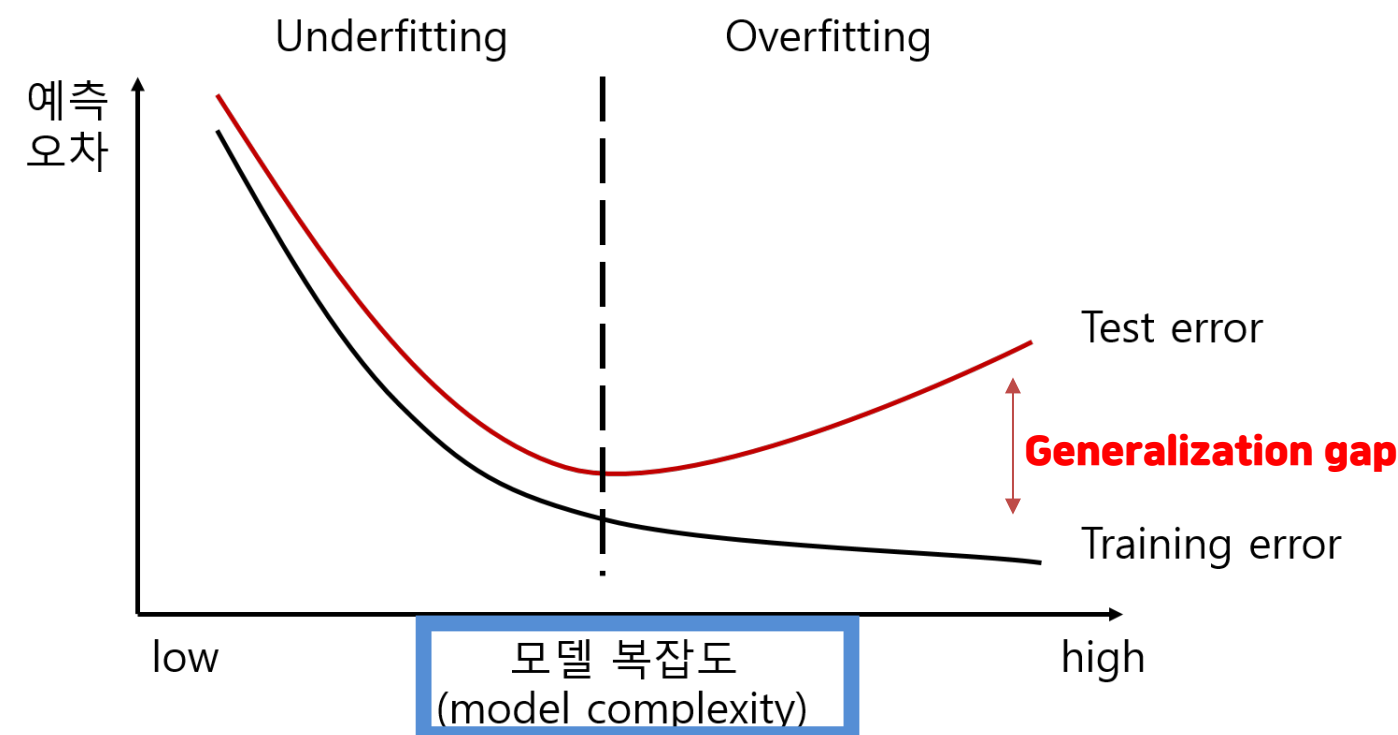
6. Batch Normalization

1. 오버피팅과 언더피팅

학습오차: 학습 데이터에 대한 오차

테스트 오차: 테스트 데이터에 대한 오차

일반화 차이(**generalization gap**): 두 오차의 차이



<https://shryu8902.github.io/working/2019-02-01-overfit/>

	학습오차 큼	학습오차 작음
일반화 차이 큼	언더피팅	오버피팅
일반화 차이 작음	언더피팅	이상적인 상태

모델의 복잡도 \approx 모델의 수용력

인공신경망에서 수용력이 높아진다는 것은 은닉층의 갯수 또는 각 은닉층의 노드의 갯수가 증가함을 의미합니다.

1. 오버피팅과 언더피팅 복습

학습을 잘 하고싶다? '학습을 가장 잘 한다'를 정의해봅시다.

어떤 모델이 학습을 가장 잘 한다: 어떤 모델의 Test accuracy가 임의의 다른 모델에 비해 높다.

모델의 Test accuracy를 높이기 위해서는 반드시 오버피팅과 언더피팅 모두 고려해야합니다.

왜? 언더피팅은 학습이 아예 이뤄지지 않은 거고, 오버피팅은 모델이 Training data에만 적합되기 때문입니다.

즉, accuracy가 쪽쪽 올라간다고 모델을 계속 복잡하게 만들다가는 Test accuracy가 50%가 나올 수 있다는 것!

강의자는 오버피팅을 언제 경험했나요?

2. 오버피팅 해결방법

2-1: Regularization(정형화)

정형화는 어떤 제약 조건을 '손실함수'에 걸어줌으로써 오버피팅을 해결하는 기법입니다.

	$\ln\lambda = -\infty$ (정형화X)	$\ln\lambda = -18$ (정형화O)	$\ln\lambda = 0$ (정형화O)
w0	0.35	0.35	0.13
w1	232.37	4.74	-0.05
w2	-5321.83	-0.77	-0.06
w3	-231639.30	-31.97	-0.05
w4	1061800.52	-3.89	-0.03
w5	-557682.99	55.28	-0.02
w6	125201.43	41.32	-0.01

Pattern Recognition and Machine Learning(Springer,2006)

$\ln\lambda = -\infty$ 의 경우
왜 이런 현상이 발생했을까요? 주어진 데이터를 정확히 맞추기 위해 파라미터 값들을 조정하다보니 이렇게 된 것입니다. 오버피팅됐을 때의 변수들을 뽑아보면 가중치의 값들이 들쭉날쭉하는 경우가 많습니다.

λ 가 무엇 일까요?

2-1: Regularization(정형화)

정형화

L1 정형화: $\omega^* = \arg\min_{\omega} SSE + \lambda \sum_{i=1}^k |w_i|$

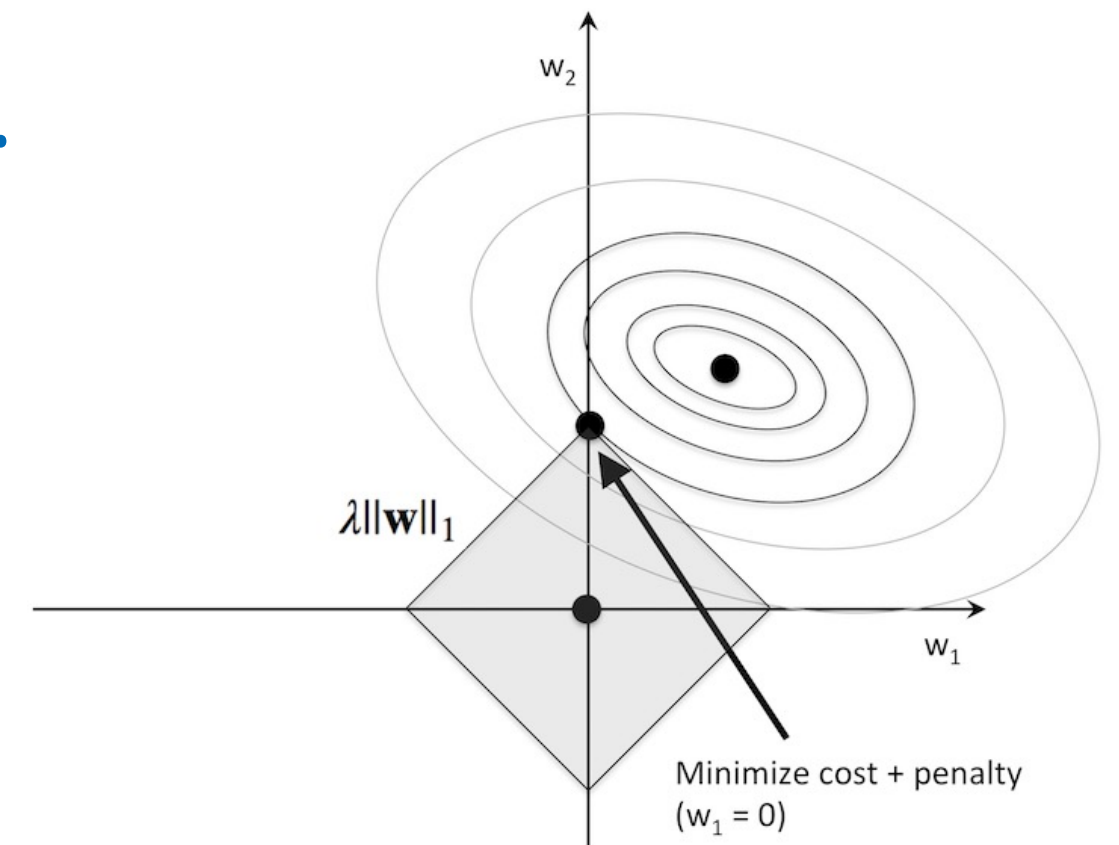
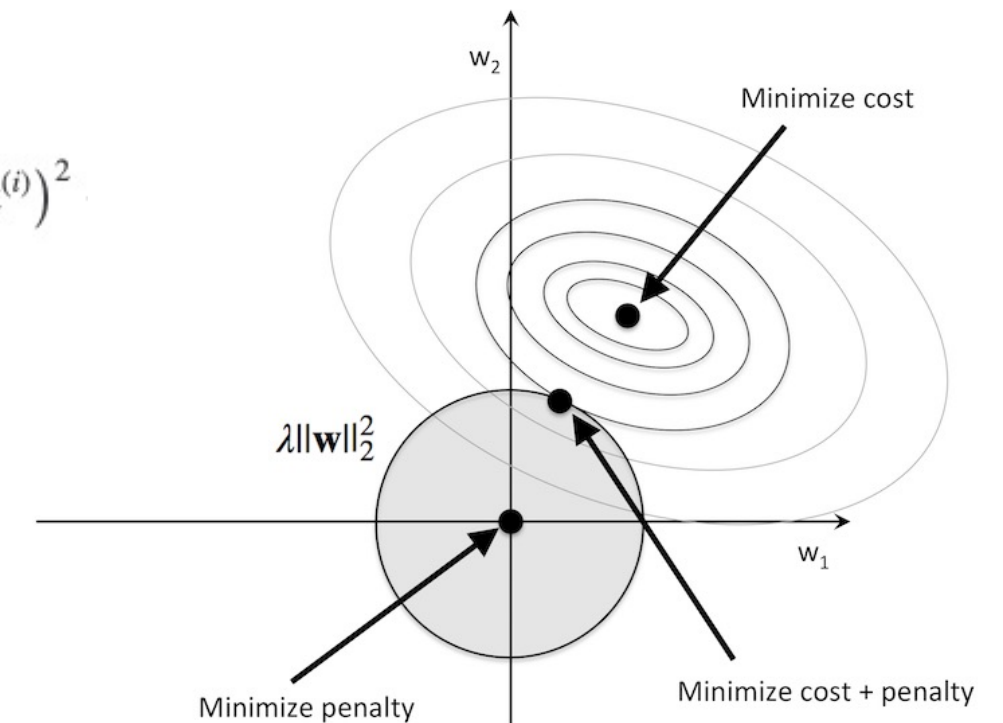
L2 정형화: $\omega^* = \arg\min_{\omega} SSE + \lambda \sum_{i=1}^k w_i^2$

L1 penalty , L2 penalty

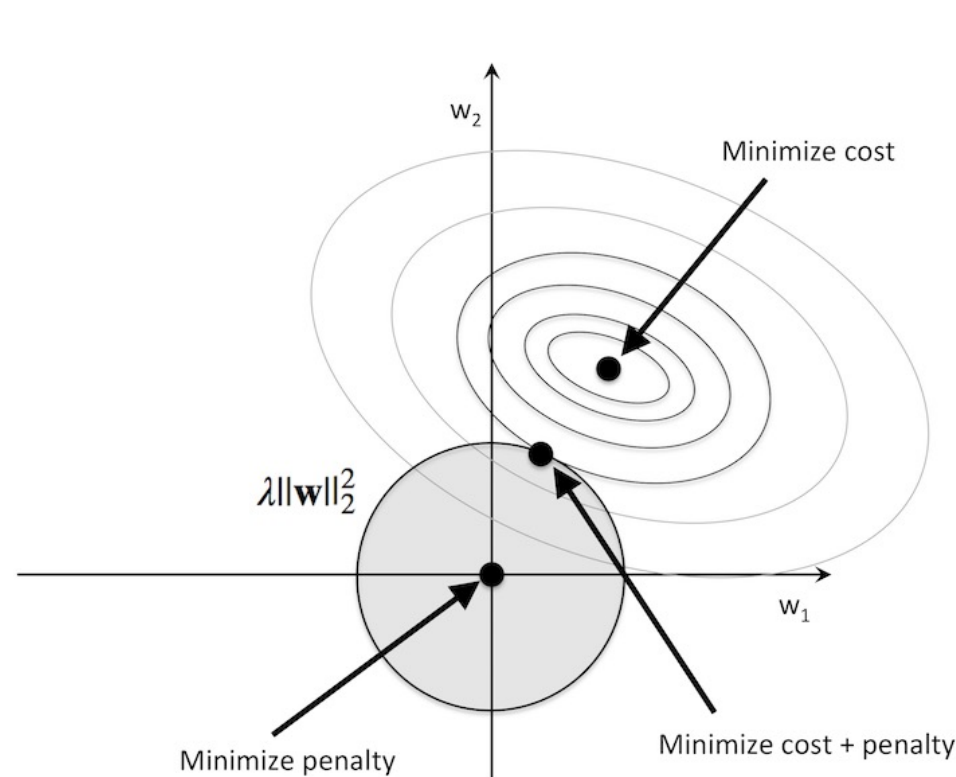
Idea: 학습은 Loss를 줄이는 게 목적입니다. λ 가 커지면 그만큼 w 의 값들이 작아지겠지요. w 의 값들이 작아진다면 함수의 형태는 단순해지고 주어진 데이터에 오버피팅하는 정도도 줄어들게 됩니다.

그림 설명

MSE 등 다른 Loss Function을 사용 하셔도 됩니다.

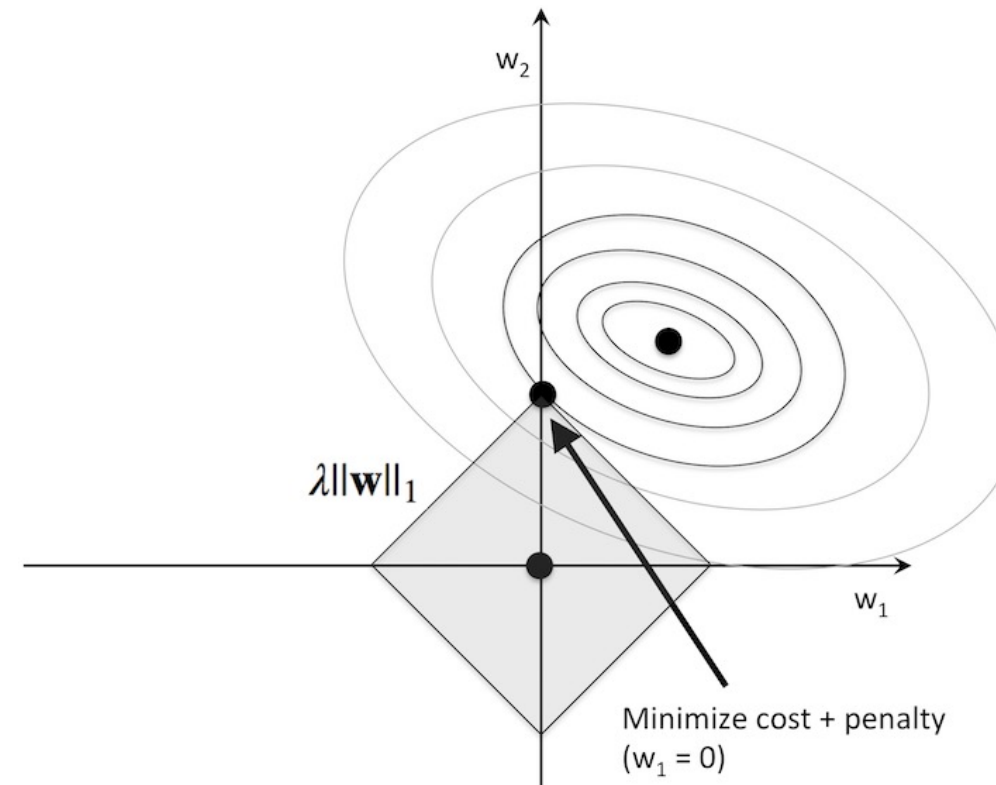
$$SSE = \sum_{i=1}^n (\text{target}^{(i)} - \text{output}^{(i)})^2$$


2-1: Regularization(정형화)



L2정형화는 수식적인 해를 구할 수 있다는 장점을 가지고 있습니다. (미분이 쉬우니까 그런 것 같습니다.)

하지만 w 가 0이 되는 부분에서 타원과 원의 접점이 있기 어려울 것 같습니다.



반대로 L1 정형화의 경우, 최적화하면 w 값들이 0이 되는 경우가 많습니다.

이 때 0이 되는 w 값은 오차를 줄이는 데에 영향이 없다고 판단되는 것이고, 이를 L1 정형화의 **특성선택**(feature selection)이라고 합니다.

SSE와 penalty항을 모두 최소화하는 부분은, 타원과 원의 접점. 타원과 사각형의 접점입니다.

2-1: Regularization(정형화)

코드로 사용

```
# 정형화는 weight_decay로 줄 수 있습니다.  
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay=0.1)
```

Pytorch는 L2 정형화만 제공합니다. 그래서 L1정형화를 사용하길 원하신다면 따로 정의를 하셔야합니다.

```
all_parameters = torch.cat([x.view(-1) for x in model.parameters()])  
#특정 부분에 제약을 걸고싶으면  
#all_parameters = torch.cat([x.view(-1) for x in model.layer.parameters()]) 이렇게 바꿔주시면 됩니다.  
lamda1=0.05  
l1_regularization = lamda1 * torch.norm(all_parameters, 1)  
l2_regularization = lamda1 * torch.norm(all_parameters, 2)  
  
loss = loss_func + l1_regularization  
loss = loss_func + l2_regularization
```

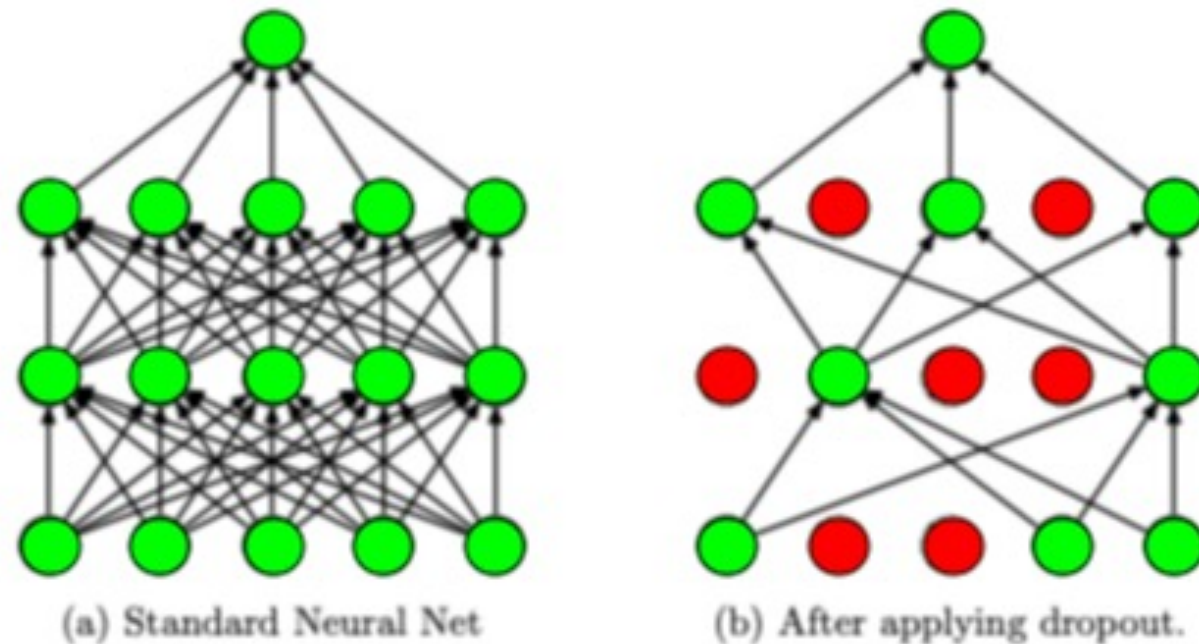

An illustration of a spiral-bound notebook with a blue cover and a white page. The spiral binding is on the left side, with blue rings. The page is mostly blank, with the text '간단하게 복습' centered in the middle. There are some faint horizontal lines at the bottom of the page.

간단하게 복습

2. 오버피팅 해결방법

2-2: Dropout

학습 때 랜덤으로 몇 개의 노드를 비활성화하여 오버피팅을 방지하는 방법!



https://cdn-images-1.medium.com/max/518/0*EY8R7nS10y5kQzOx

학습이 진행될 때 확률 p 로 특정 노드를 비활성화 하는 방법입니다.

특정 노드를 비활성화 하면 이전 층에서 계산된 값들이 그 노드에 전달되지 않습니다. 이렇게 하면 모델의 수용력은 낮아지게 됩니다.

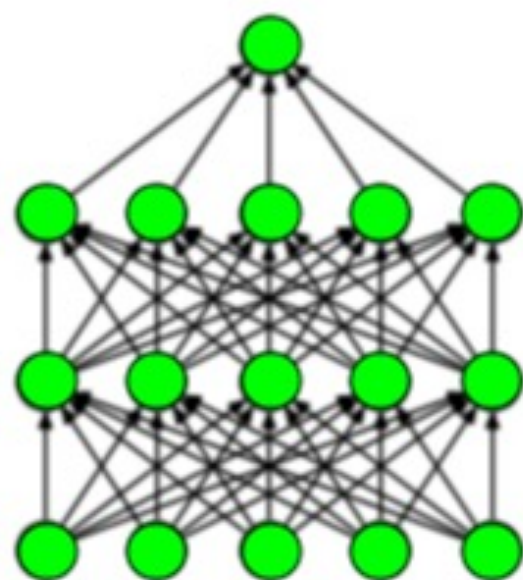
그런데.. 이렇게 하면 학습이 아예 안 될 것 같은데.. 어떻게 이게 가능한 건가요?

드롭아웃은 수용력이 낮은 모델들의 앙상블 개념으로도 볼 수 있습니다. 학습 때 만들어진 여러 개의 '낮은 수용력을 가진 모델' 들이 테스트할 때에는 드롭아웃을 하지 않음으로써 합쳐지기 때문입니다.

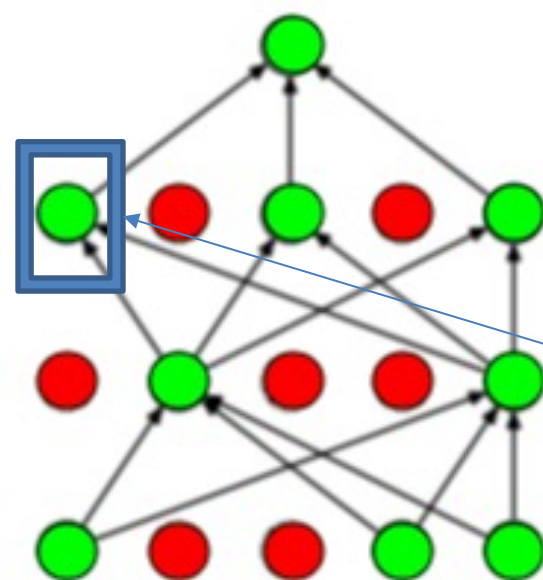
2-2: Dropout

테스트할 때 모든 뉴런의 값을 전달한다고 했는데, 그렇게 하면, 기존에 전달되던 값보다 훨씬 더 큰 값이 다음 노드에 전달되므로 문제가 발생할 것입니다. 그래서 테스트 할 때에는 드롭아웃을 전달되는 값에 곱해줍니다.

예시



(a) Standard Neural Net



(b) After applying dropout.

$P=0.5$

비활성화가 가능한 뉴
런 수 : 15

이 뉴런에 대하여.

2-2: Dropout

코드 사용

드롭아웃을 중간중간에 넣어줌으로써 모델이 오버피팅하는 경우 이를 어느정도 극복할 수 있습니다.
오버피팅하지 않는 상태에서 정형화나 드롭아웃을 넣으면 오히려 학습이 잘 안됩니다.

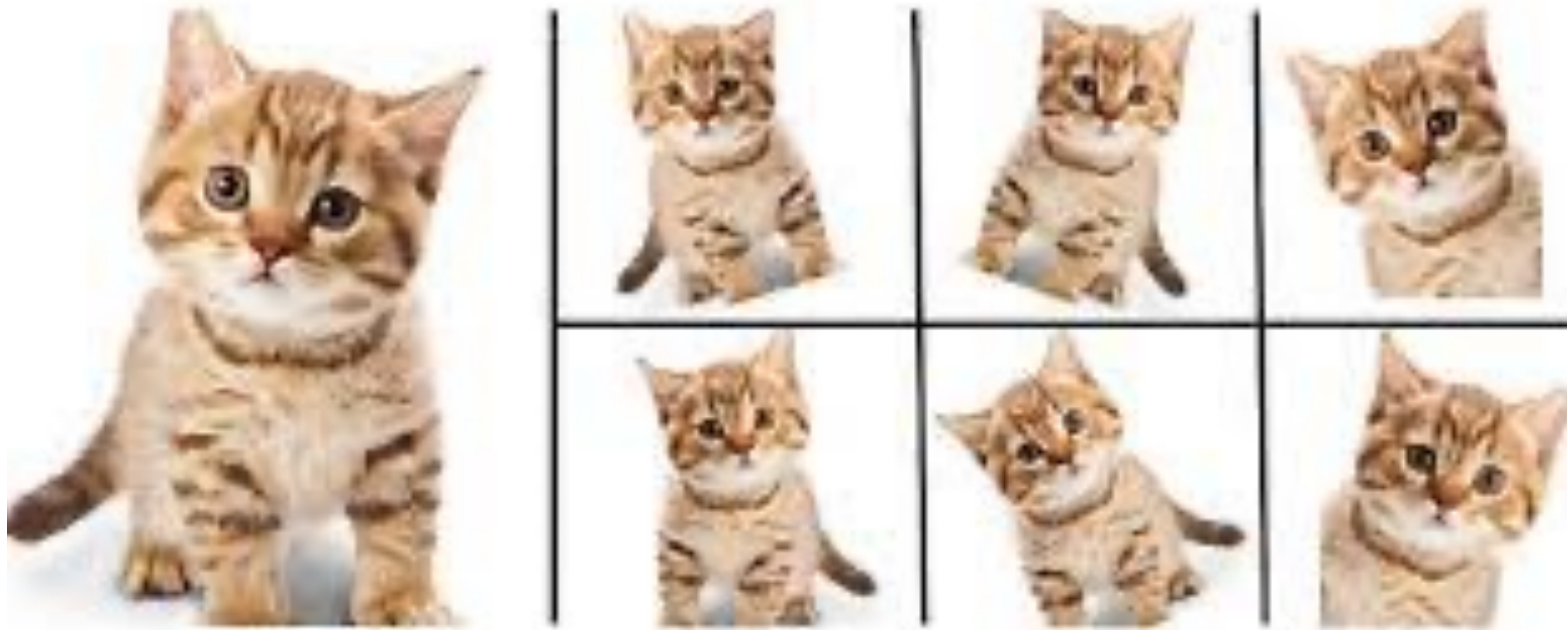
```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer = nn.Sequential(
            nn.Conv2d(1, 16, 3, padding=1), # 28
            nn.ReLU(),
            #nn.Dropout2d(0.2),
            nn.Conv2d(16, 32, 3, padding=1), # 28
            nn.ReLU(),
            #nn.Dropout2d(0.2),
            nn.MaxPool2d(2, 2), # 14
            nn.Conv2d(32, 64, 3, padding=1), # 14
            nn.ReLU(),
            #nn.Dropout2d(0.2),
            nn.MaxPool2d(2, 2) # 7
        )
        self.fc_layer = nn.Sequential(
            nn.Linear(64*7*7, 100),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(100, 10)
        )
```

A spiral binding on the left side of a notebook page, consisting of a series of blue and white rings.

간단하게 복습

**다음 장부터는 학습을 더 잘하는
방법에 대한 내용이 진행됩니다.**

3. Data Augmentation(데이터 증강)



단순히 기존의 데이터를 돌리고,
뒤집어서 학습 데이터의 양을 증
가시키는 방법.

Enlarge your Dataset

<https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/>

```
[ ] mnist_train = dset.MNIST("./", train=True,  
                             transform = transforms.Compose([  
                                 transforms.Resize(34),  
                                 transforms.CenterCrop(28),  
                                 transforms.RandomHorizontalFlip(),  
                                 transforms.Lambda(lambda x: x.rotate(90)),  
                                 transforms.ToTensor(),
```

원래 28x28인 이미지를 34x34로 늘립니다.
중앙 28x28를 뽑아냅니다.
랜덤하게 좌우반전 합니다.
람다함수를 이용해 90도 회전해줍니다.
이미지를 텐서로 변형합니다.

4. 가중치 초기화

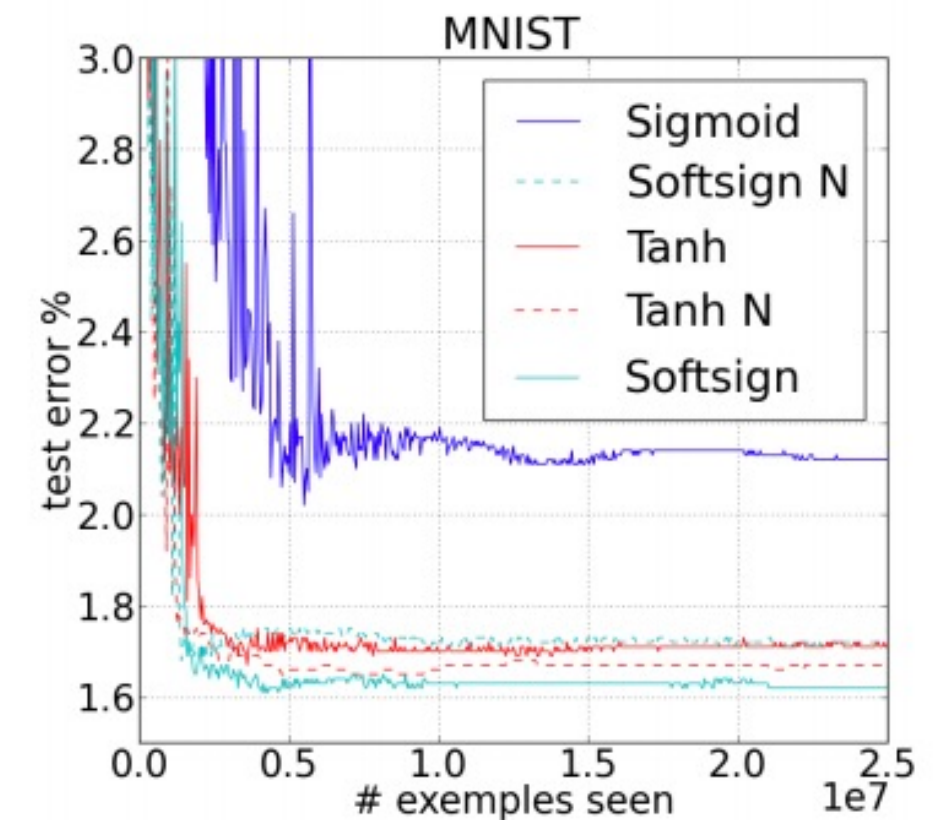
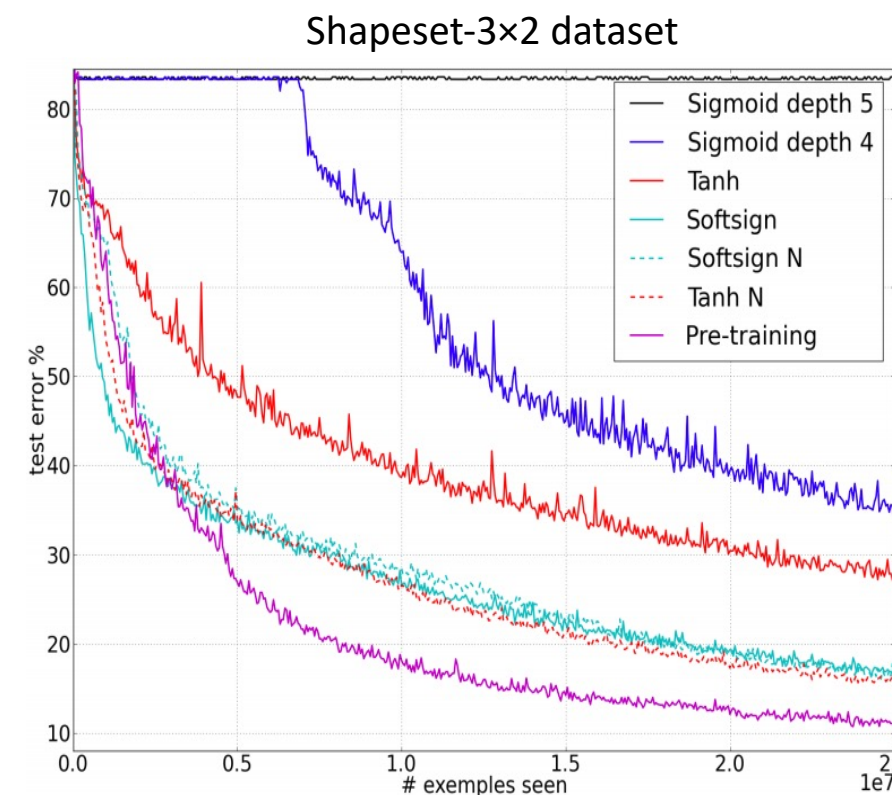
어떤 값에서 시작하는 게 학습을 더 효율적으로 바꿀까?라는 질문에서 시작합니다.

초기화의 대표적인 방법에는 **Xavier Glorot 초기화**, **Kaiming He 초기화**가 있습니다.

(1) Xavier Glorot 초기화

가중치의 초기값을
 $N(0, \text{var}=2/(n_{in} + n_{out}))$
에서 추출하는 게 핵심!

이렇게 초기값을 설정하면 데이터가 레이어를 계속 통과하더라도 활성화 값이 일정 범위 내에 있게 됩니다. 활성화 값이 너무 커지거나 너무 작아지지 않는 것이지요.



4. 가중치 초기화

(2) Kaiming He 초기화

Xavi 가중치 논문이 발표된 이후, 렐루활성화 함수가 많이 쓰이게 되었습니다. 이에 맞춰 만들어진 것이 He 초기화입니다. He 초기화는 Relu활성화 함수를 쓸 때 쓰면 학습에 좋습니다.

He 초기화는 가중치를 $N(0, \frac{2}{(1+a^2) \times n_{in}})$ 에서 추출합니다.

렐루를 사용함을 디폴트로 두기 때문에, a의 디폴트는 0입니다. Leaky ReLU를 사용하게 되면 그것의 기울기를 a에 대입합니다.

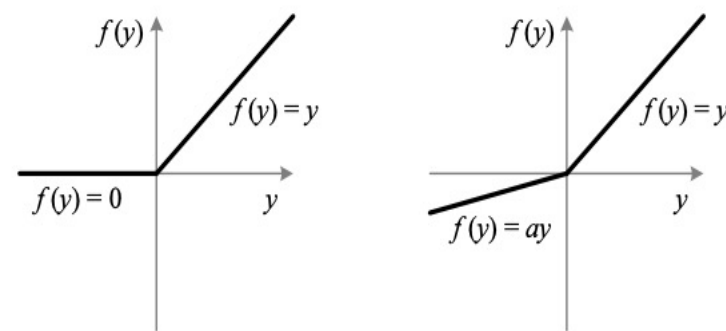


Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.

<https://arxiv.org/pdf/1502.01852.pdf>

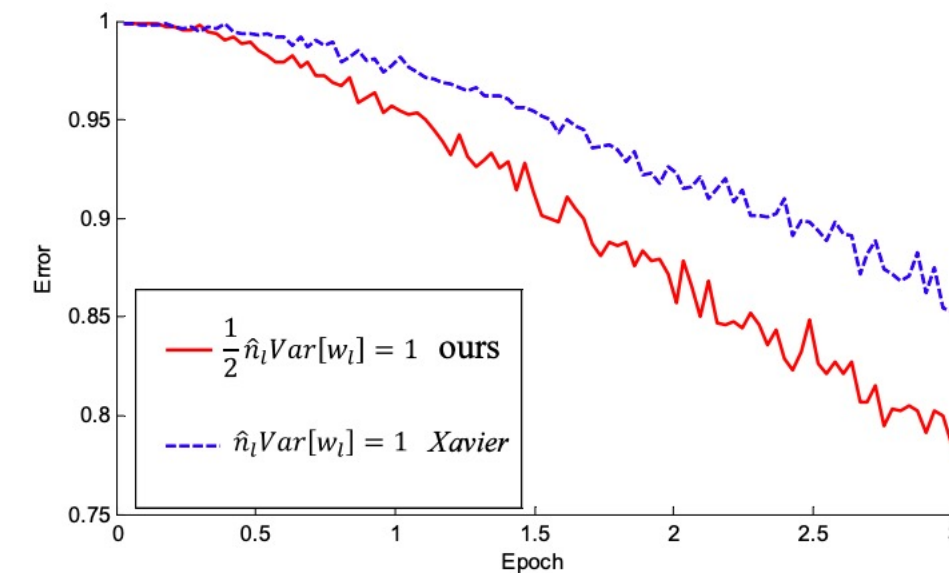


Figure 2. The convergence of a **22-layer** large model (B in Table 3). The x-axis is the number of training epochs. The y-axis is the top-1 error of 3,000 random val samples, evaluated on the center crop. We use ReLU as the activation for both cases. Both our initialization (red) and “Xavier” (blue) [7] lead to convergence, but ours starts reducing error earlier.

<https://arxiv.org/pdf/1502.01852.pdf>

4. 가중치 초기화

코드사용

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer = nn.Sequential(
            nn.Conv2d(1, 16, 3, padding=1), # 28 x 28
            nn.ReLU(),
            nn.Conv2d(16, 32, 3, padding=1), # 28 x 28
            nn.ReLU(),
            nn.MaxPool2d(2, 2), # 14 x 14
            nn.Conv2d(32, 64, 3, padding=1), # 14 x 14
            nn.ReLU(),
            nn.MaxPool2d(2, 2) # 7 x 7
        )

        self.fc_layer = nn.Sequential(
            nn.Linear(64*7*7, 100),
            nn.ReLU(),
            nn.Linear(100, 10)
        )

        # 초기화 하는 방법
        # 모델의 모듈을 차례대로 불러옵니다.
        for m in self.modules():
            # 만약 그 모듈이 nn.Conv2d인 경우
            if isinstance(m, nn.Conv2d):
                ...

            # 작은 숫자로 초기화하는 방법
            # 가중치를 평균 0, 편차 0.02로 초기화합니다.
            # 편차를 0으로 초기화합니다.
```

```
# 작은 숫자로 초기화하는 방법
# 가중치를 평균 0, 편차 0.02로 초기화합니다.
# 편차를 0으로 초기화합니다.
m.weight.data.normal_(0.0, 0.02)
m.bias.data.fill_(0)

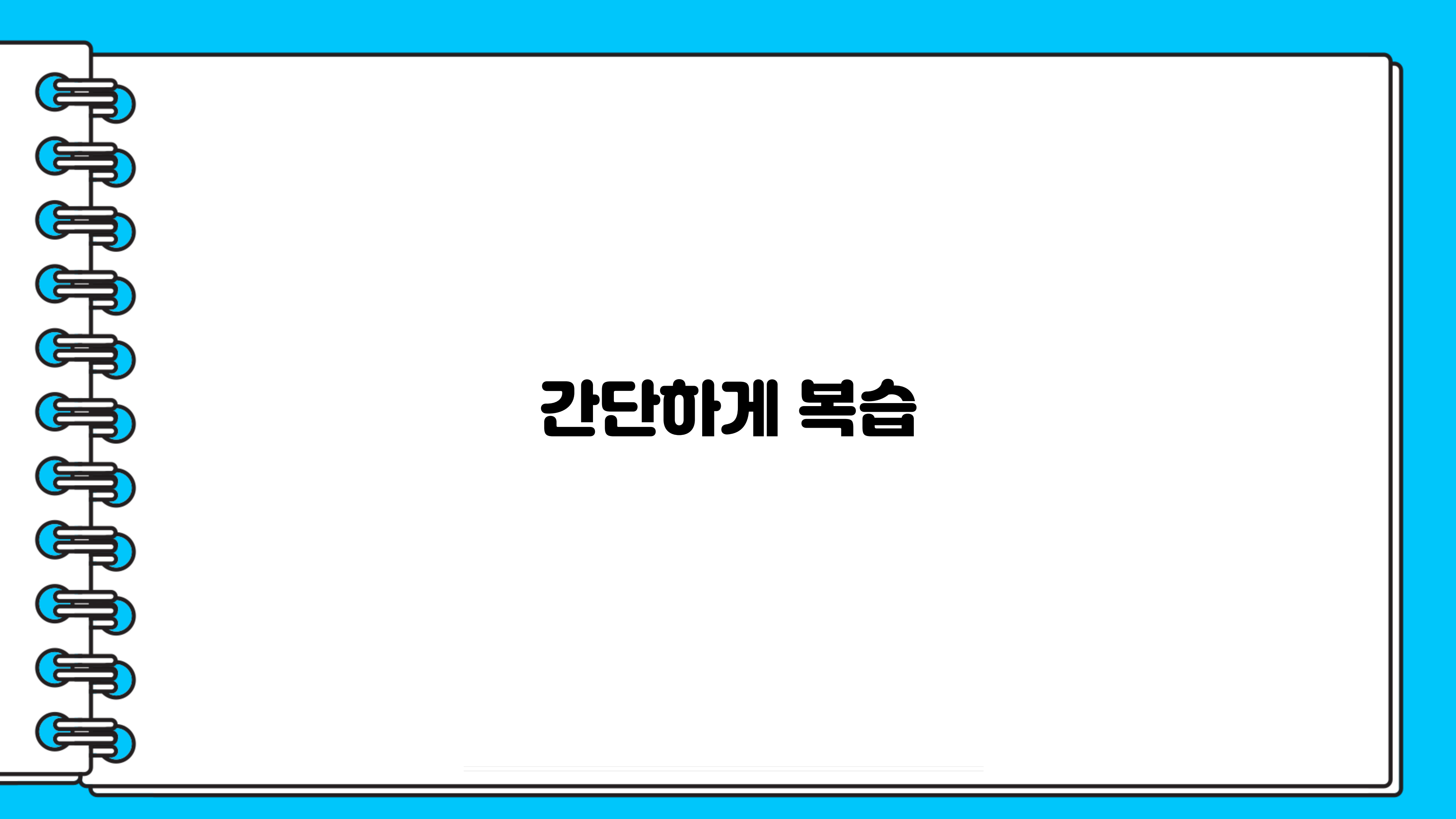
# Xavier Initialization
# 모듈의 가중치를 xavier normal로 초기화합니다.
# 편차를 0으로 초기화합니다.
init.xavier_normal(m.weight.data)
m.bias.data.fill_(0)
'''

# Kaming Initialization
# 모듈의 가중치를 kaming he normal로 초기화합니다.
# 편차를 0으로 초기화합니다.
init.kaiming_normal_(m.weight.data)
m.bias.data.fill_(0)

# 만약 그 모듈이 nn.Linear인 경우
elif isinstance(m, nn.Linear):
    ...

    # 작은 숫자로 초기화하는 방법
    # 가중치를 평균 0, 편차 0.02로 초기화합니다.
    # 편차를 0으로 초기화합니다.
    m.weight.data.normal_(0.0, 0.02)
    m.bias.data.fill_(0)

    # Xavier Initialization
    # 모듈의 가중치를 xavier normal로 초기화합니다.
```

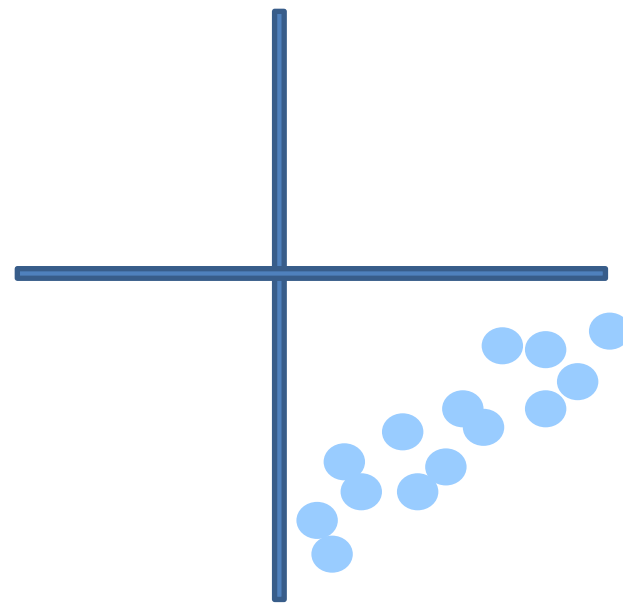
An illustration of a spiral-bound notebook with a blue cover and a white page. The spiral binding is on the left side, with blue rings. The page is mostly blank, with the text '간단하게 복습' centered in the middle. There are some faint horizontal lines at the bottom of the page.

간단하게 복습

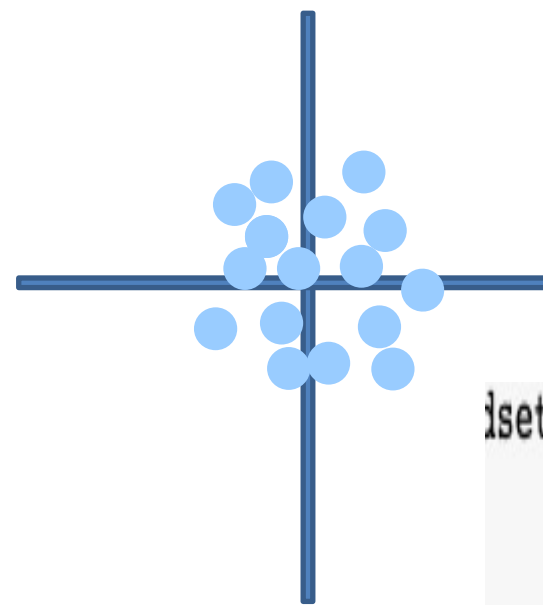
5. 입력데이터 정규화

정규화는 학습시 들어오는 입력값이 일정한 분포로 들어오면 더 효율적인 학습을 할 수 있다는 아이디어에 기반했습니다.

정규화에는 크게 표준화(standardization)와 최소극대화(minmax) 방법이 있습니다.



정규화 이전



$$\frac{x - \mu}{\sigma}$$

```
dset.MNIST("./", train=True,  
            transform=transforms.Compose([  
                transforms.ToTensor(),  
                transforms.Normalize(mean=(0.1307,), std=(0.3081,))  
            ]),
```

```
dset.MNIST("./", train=True,  
            transform=transforms.Compose([  
                transforms.ToTensor(),  
                transforms.Normalize(mean=(0.485, 0.456, 0.406), #channel이 세 개인 경우  
                                     std=(0.229, 0.224, 0.225))  
            ])
```

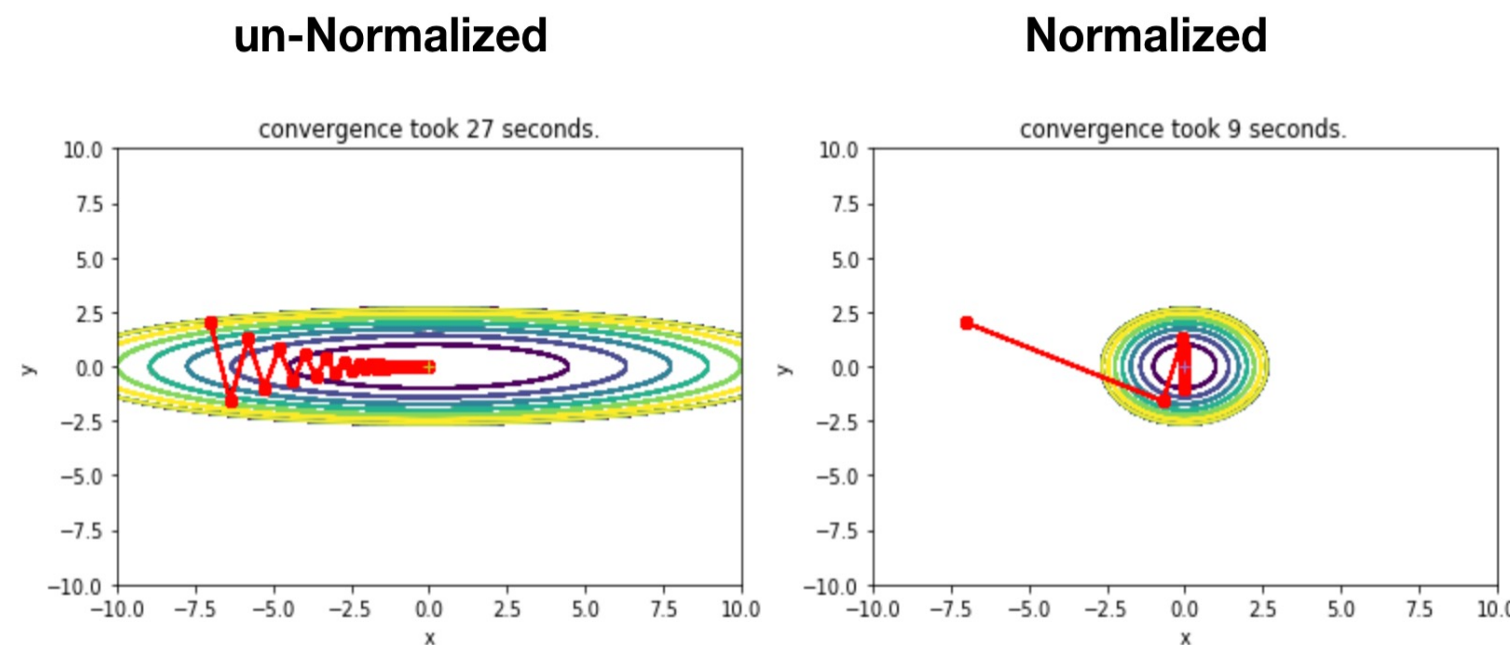
5. 입력 데이터 정규화

최소극대화 mimax 정규화

데이터를 0에서 1 사이로 압축하거나 늘리는 방법

```
x = (x - x.min()) / (x.max() - x.min())
```

너무나 작은 값이 있거나, 너무나 큰 값이 있는 경우에는 오히려 학습에 방해가 되니 주의!



데이터가 정규화되지 않았다면, 데이터의 범위가 변수마다 다를 것입니다. 때문에 학습할 때 어떤 변수에서는 빠르게, 어떤 변수에서는 느리게 학습을 해야하는데, 그럴 수 없습니다. 따라서 모든 변수의 범위를 일정하게 바꿔서 비교적 빠른 학습률을 꼭 적용할 수 있고 이에 따라 최소 Loss 지점에 더욱 빠르게 도착할 수 있습니다.

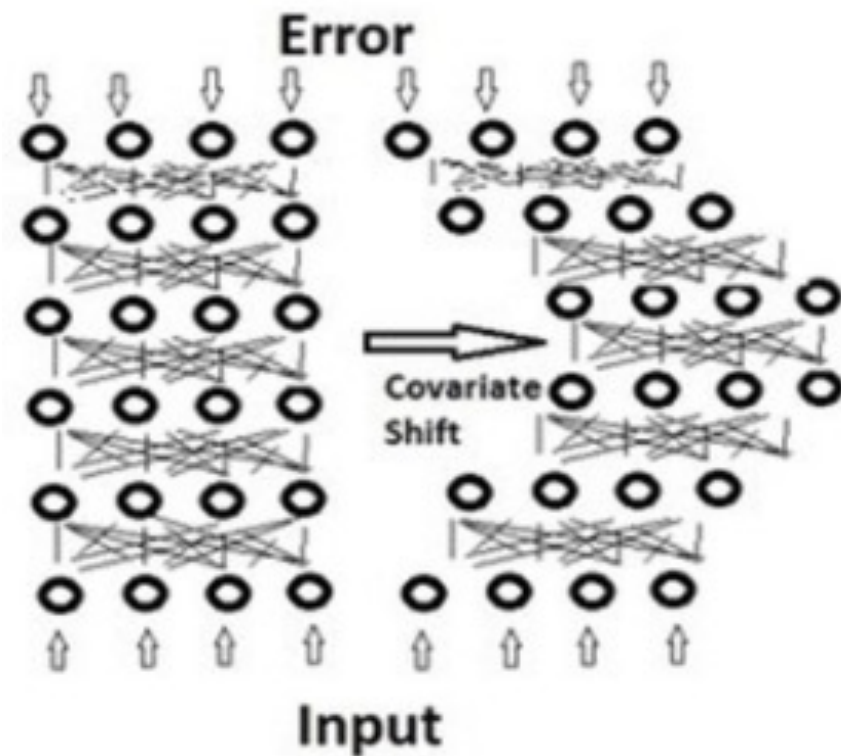
A spiral binding on the left side of a notebook page, consisting of a series of blue and white rings.

간단하게 복습

입력값에만 적용 할 필요는 없습니다!

6. Batch Normalization (배치 정규화)

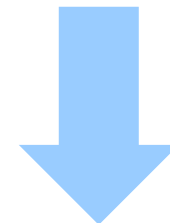
공변량 변화 (covariate shift): 하나의 신경망에 대해서 입력의 범위가 바뀌는 것.



<https://deep-learning-study.tistory.com/421>

가중치 학습이 비효율적

층이 깊어질수록 학습의 비효율이 극대



배치 정규화: 한 번에 입력으로 들어오는 배치 단위로 정규화

배치 정규화는 보통 합성곱 연산이나, 선형변환 연산 사이에 들어갑니다. 은닉층 단위마다 배치 정규화가 들어가는 거죠.

Model	Resolution	Crops	Models	Top-1 error	Top-5 error
GoogLeNet ensemble	224	144	7	-	6.67%
Deep Image low-res	256	-	1	-	7.96%
Deep Image high-res	512	-	1	24.88	7.42%
Deep Image ensemble	variable	-	-	-	5.98%
BN-Inception single crop	224	1	1	25.2%	7.82%
BN-Inception multicro	224	144	1	21.99%	5.82%
BN-Inception ensemble	224	144	6	20.1%	4.9%*

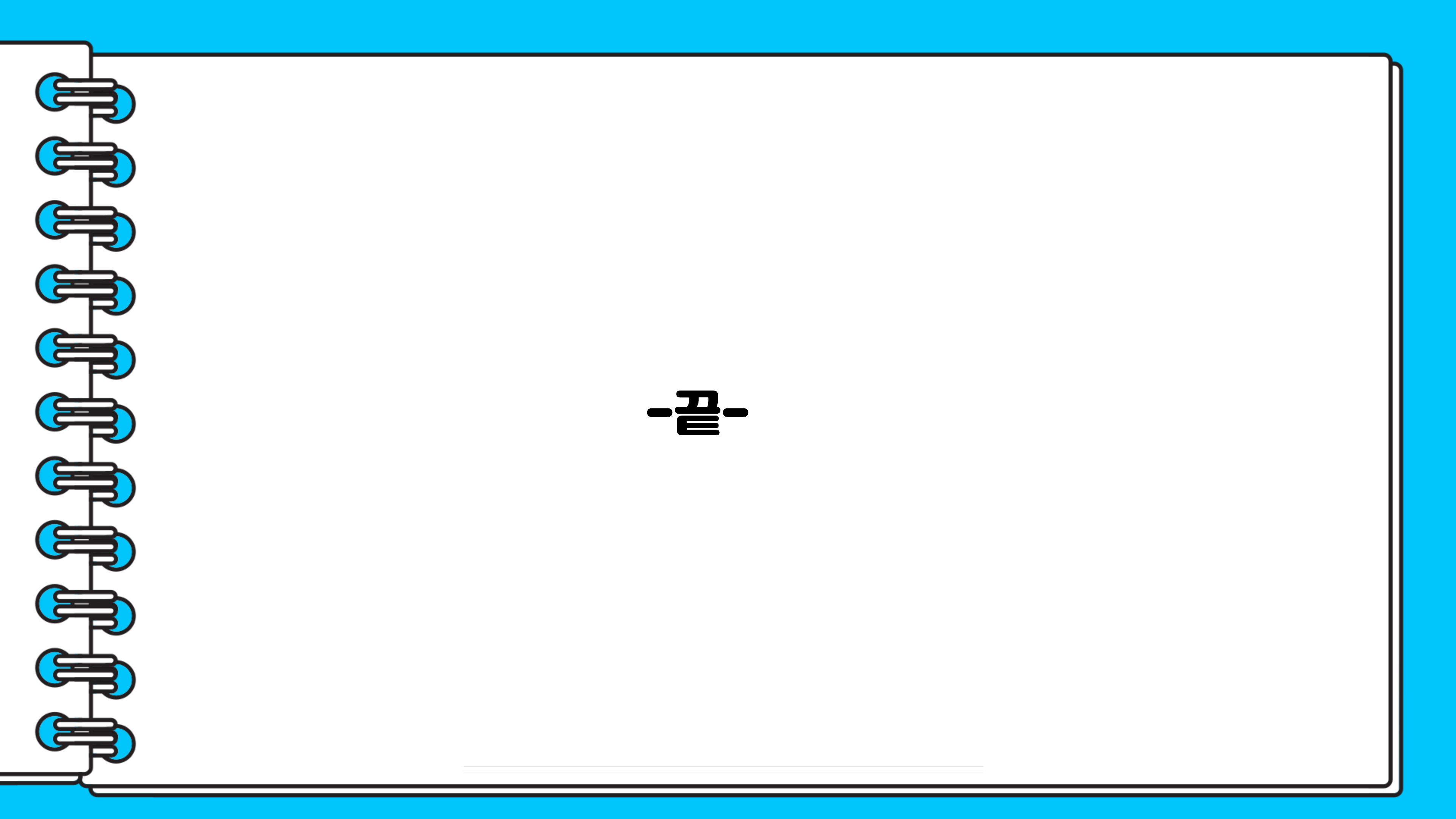
<https://arxiv.org/pdf/1502.03167.pdf>

6. Batch Normalization (배치 정규화)

코드 사용

```
# 입력 데이터를 정규화하는것처럼 연산을 통과한 결과값을 정규화할 수 있습니다.  
# 그 다양한 방법중에 대표적인것이 바로 Batch Normalization이고 이는 컨볼루션 연산처럼 모델에 한 층으로 구현할 수 있습니다.  
# https://pytorch.org/docs/stable/nn.html?highlight=batchnorm#torch.nn.BatchNorm2d  
# nn.BatchNorm2d(x)에서 x는 입력으로 들어오는 채널의 개수입니다.
```

```
class CNN(nn.Module):  
    def __init__(self):  
        super(CNN, self).__init__()  
        self.layer = nn.Sequential(  
            nn.Conv2d(1, 16, 3, padding=1), # 28 x 28  
            nn.BatchNorm2d(16),  
            nn.ReLU(),  
            nn.Conv2d(16, 32, 3, padding=1), # 28 x 28  
            nn.BatchNorm2d(32),  
            nn.ReLU(),  
            nn.MaxPool2d(2, 2), # 14 x 14  
            nn.Conv2d(32, 64, 3, padding=1), # 14 x 14  
            nn.BatchNorm2d(64),  
            nn.ReLU(),  
            nn.MaxPool2d(2, 2) # 7 x 7  
        )  
        self.fc_layer = nn.Sequential(  
            nn.Linear(64*7*7, 100),  
            nn.BatchNorm1d(100),  
            nn.ReLU(),  
            nn.Linear(100, 10)  
        )
```

12