Training Neural Network (Part1)

최하경

목차

01. Reviews

1.1 CNN

Convolutional Neural Networks

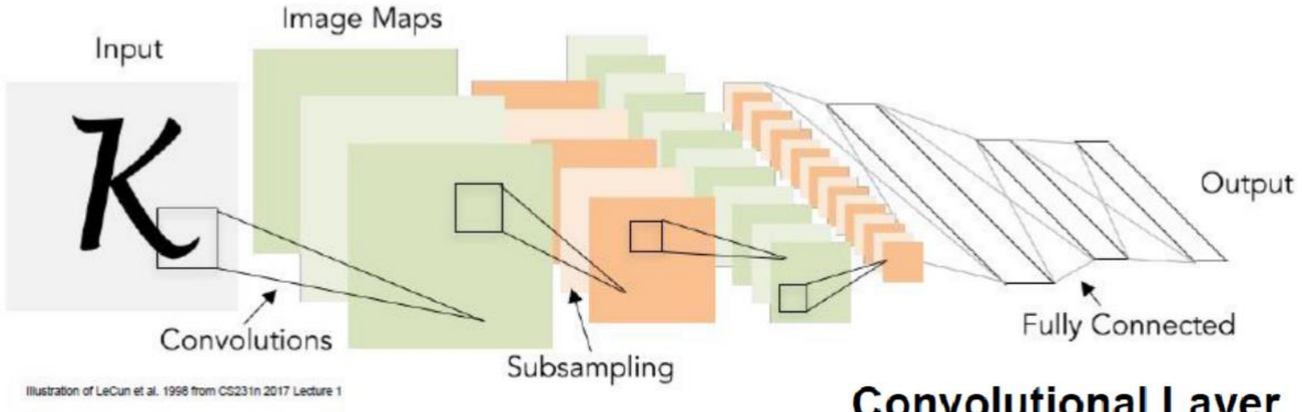
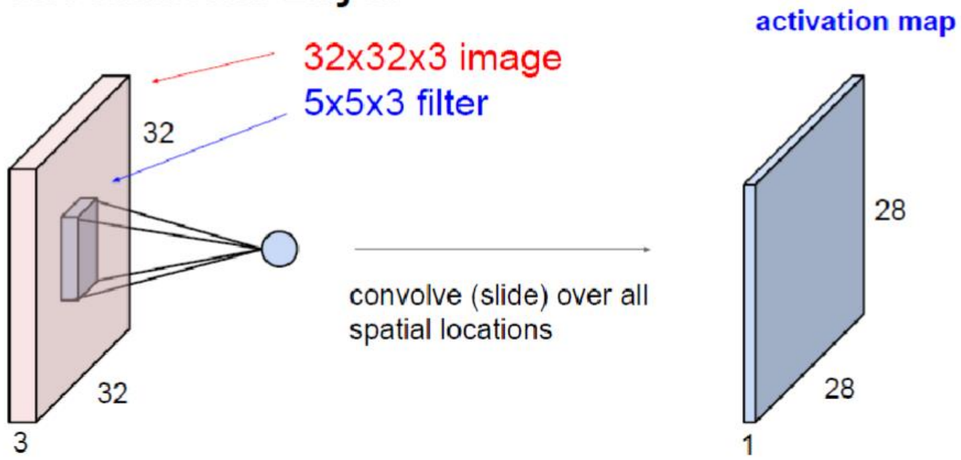


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

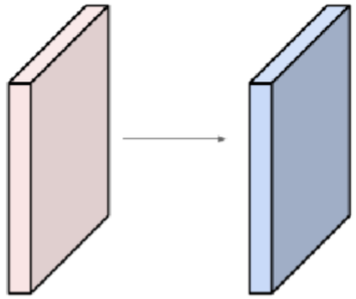
Convolutional Layer



1.1 CNN

Examples time:

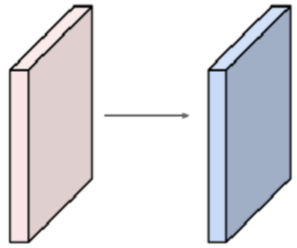
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

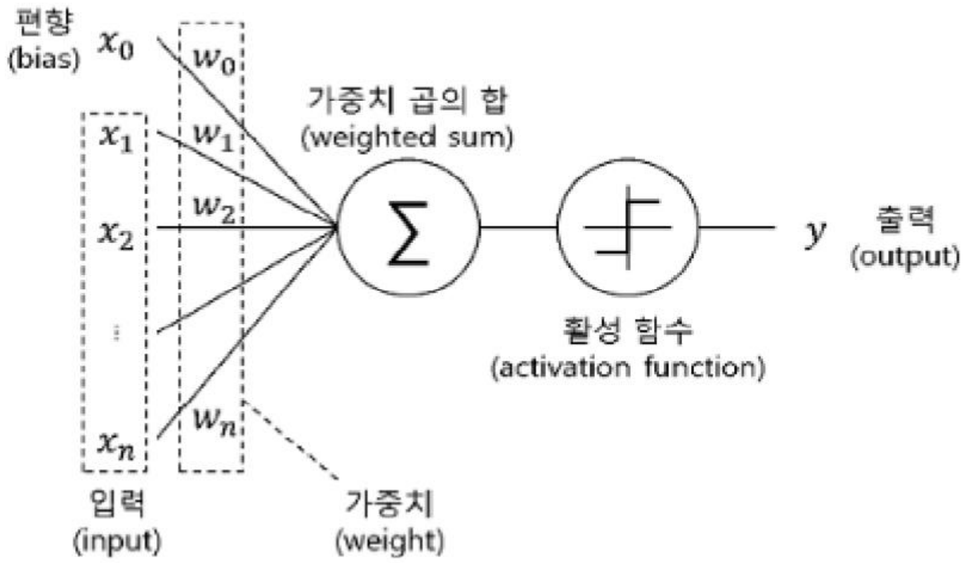
1.1 CNN

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

02. Activation Functions

2.1 Activation Functions



신경망의 학습 절차

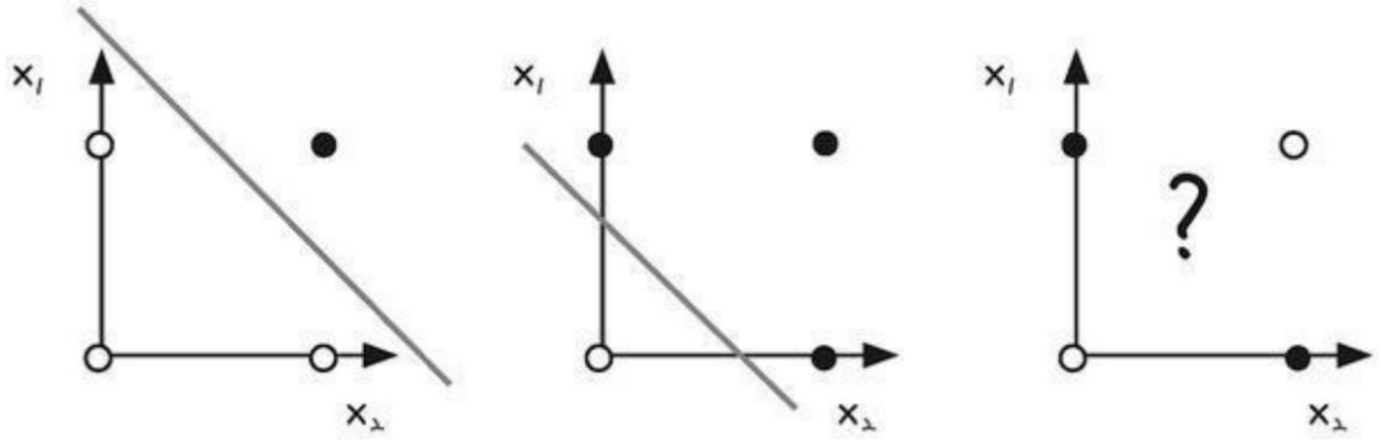
활성화 함수 :

노드의 입력값이 임계치를 넘어가면 활성화가 되고 넘지 않으면 비활성화하게끔 되어있다.

(스위치 같은 개념!)

2.1 Activation Functions

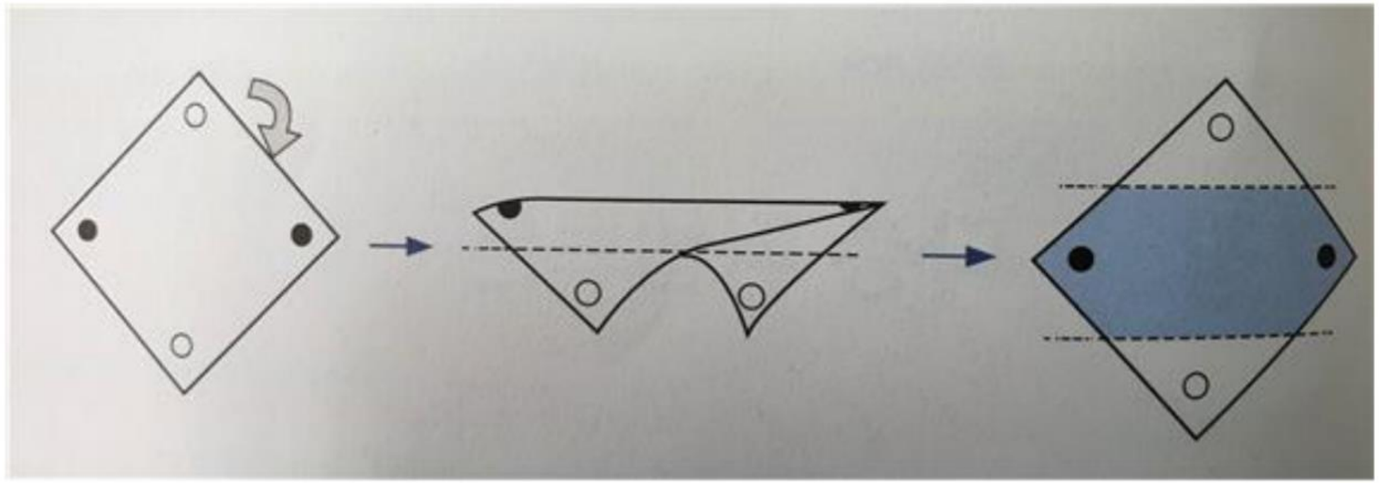
Q. 활성화 함수를 왜 사용하는가?



퍼셉트론의 한계

2.1 Activation Functions

Q. 활성화 함수를 왜 사용하는가?



다층 퍼셉트론

즉, 다층 퍼셉트론에서 모델의 표현력을 증가시키기 위해 활성화 함수가 필요하다.

2.1 Activation Functions

Q. 활성화 함수를 왜 사용하는가?

활성화 함수는 선형 함수를 비선형 함수로 출력하고 신호를 전달하는 역할을 함.

예를 들어, 활성화 함수를 h(x) = cx 선형 함수 라고 하자.

3층으로 구성된 신경망을 만들고 싶을 때, h(h(h(x))) = c • c • c • x 이다.

C^3 • x 인데 초기의 h(x) 에 a = c^3 넣은 것과 같고 선형 함수 이다.

→ 선형 함수의 퍼셉트론 즉, 1층 구조로 Layers 쌓은 것과 같다.

2.1 Activation Functions

Q. 활성화 함수를 왜 사용하는가?

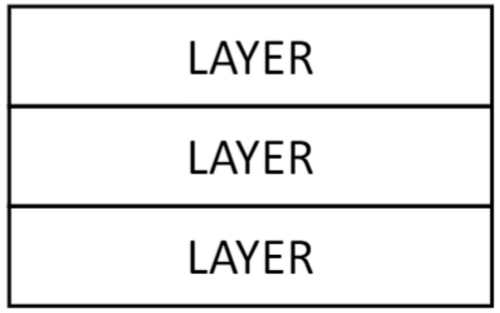
활성화 함수는 선형 함수를 비선형 함수로 출력하고 신호를 전달하는 역할을 함.

예를 들어, 활성화 함수를 h(x) = cx 선형 함수 라고 하자.

3층으로 구성된 신경망을 만들고 싶을 때, h(h(h(x))) = c • c • c • x 이다.

C^3 • x 인데 초기의 h(x) 에 a = c^3 넣은 것과 같고 선형 함수 이다.

→ 선형 함수의 퍼셉트론 즉, 1층 구조로 Layers 쌓은 것과 같다.



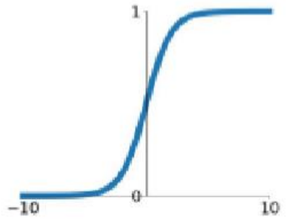
다층 퍼셉트론

단층 퍼셉트론

2.1 Activation Functions

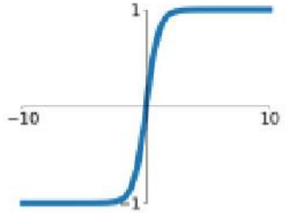
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



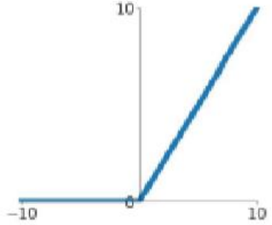
tanh

$$\tanh(x)$$



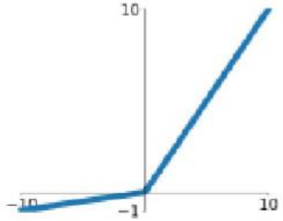
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

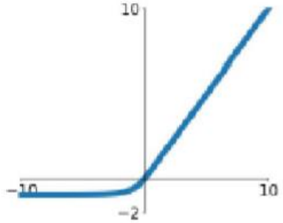


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

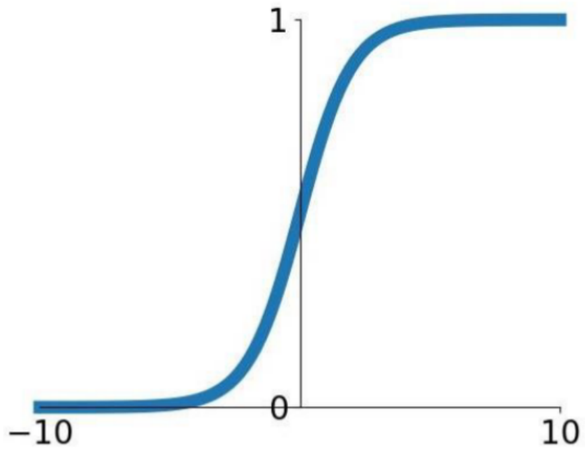
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



2.2 Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

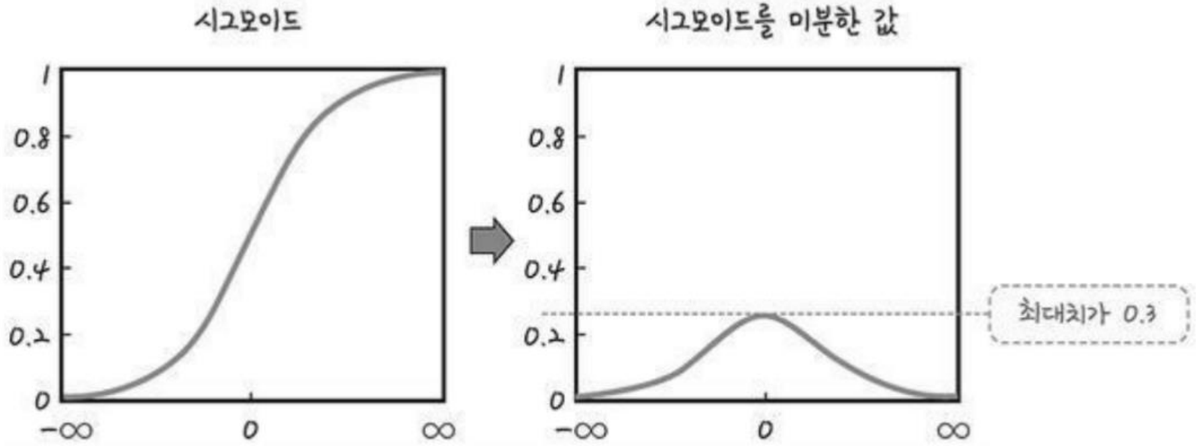


Sigmoid

특징	- Squash function - Logistic function - Historically popular since they have nice interpretation as a saturating “firing rate” of neuron
문제	- exp() -> expensive computation - Vanishing gradient - Not zero-centered

2.2 Sigmoid

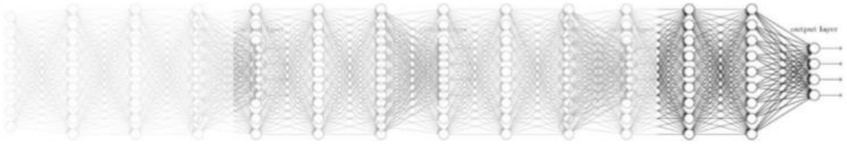
시그모이드 함수의 문제점. (Vanishing gradient)



Vanishing gradient (NN winter2: 1986-2006)

→ 시그모이드의 미분값의 최대치가 0.3 이다. 1보다 작으므로 역전파를 진행할 때 값이 0에 가까워지고 가중치를 수정하기가 어려워짐.

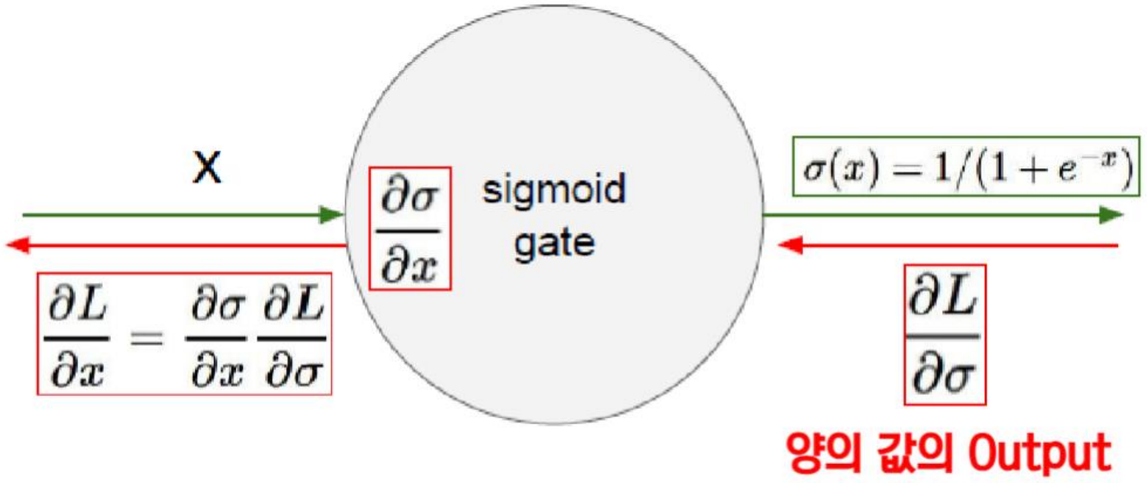
(Vanishing gradient 발생)



2.2 Sigmoid

시그모이드 함수의 문제점. (Not zero-centered)

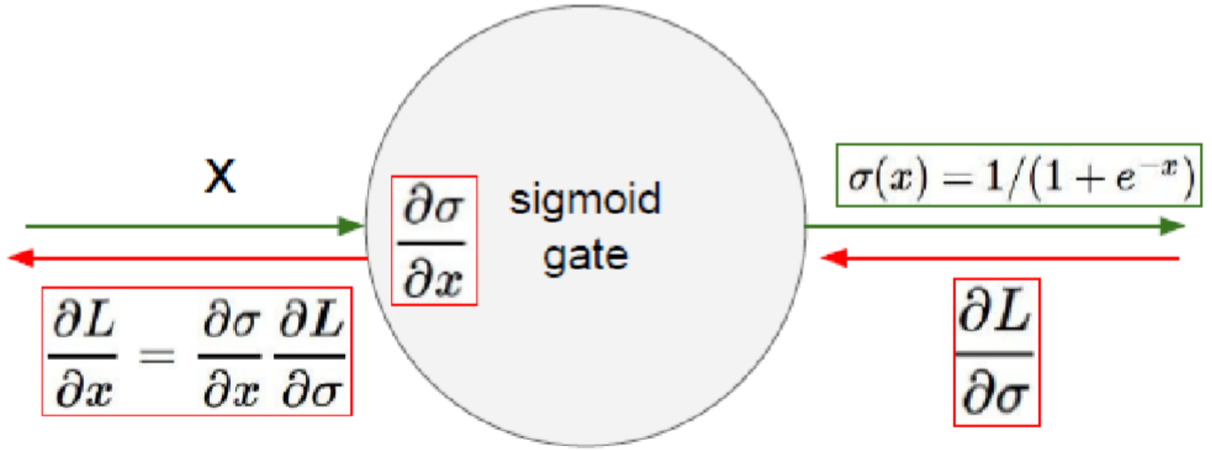
Sigmoid 는 [0,1] 범위의 output 을 출력한다.



2.2 Sigmoid

시그모이드 함수의 문제점. (Not zero-centered)

Sigmoid 는 [0,1] 범위의 output 을 출력한다.
역전파를 진행할 때, W의 gradient 는 모두 양수 또는 음수의 값을 갖게 된다.

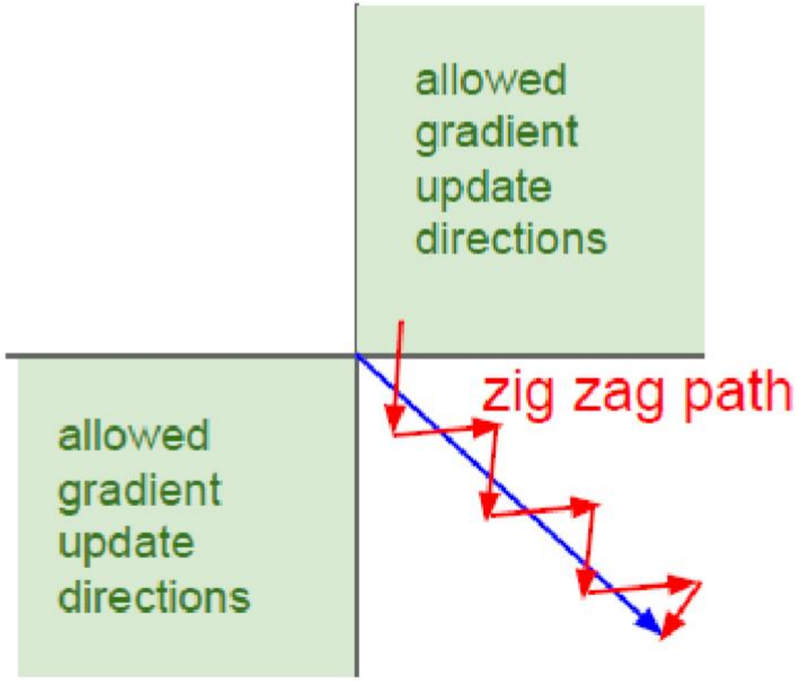


W의 grad 는 da/dx 에 의존 (upstream grad)

양의 값의 Output

2.2 Sigmoid

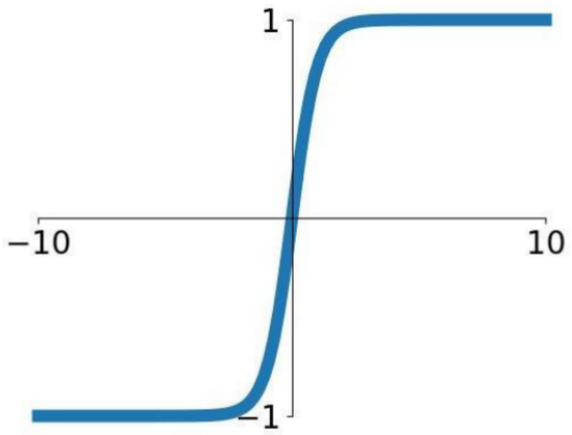
시그모이드 함수의 문제점. (Not zero-centered)



모두 양수 또는 음수이므로 zig zag 방향으로 가기 때문에 수렴이 매우 느려진다.

Not zero centered 는 수렴이 느릴 수 밖에 없다.

2.3 tanh



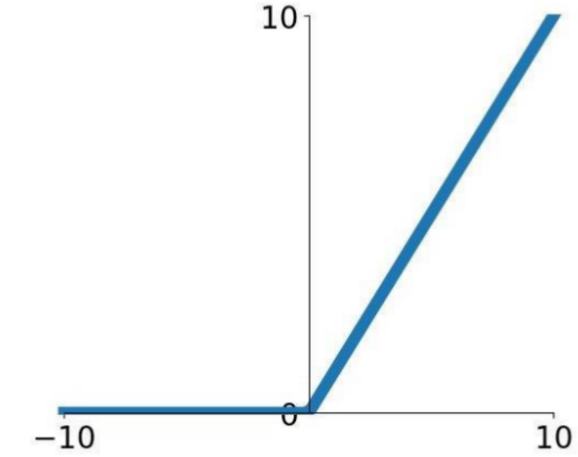
tanh(x)

특징	- Squash numbers to range [-1, 1] - Zero centered
문제	- Vanishing gradient

2.4 ReLU

Sigmoid 함수의 단점

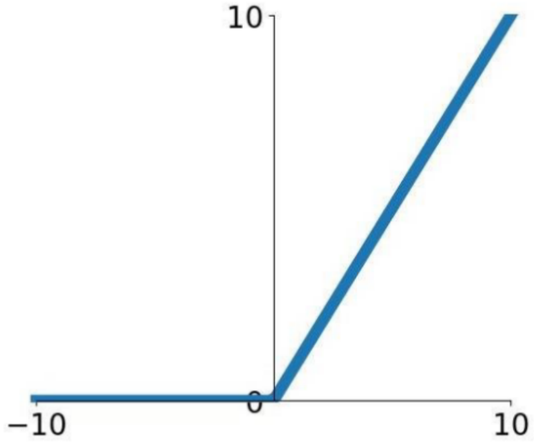
1. Vanishing gradient
2. Slow convergence



ReLU
(Rectified Linear Unit)

2.4 ReLU

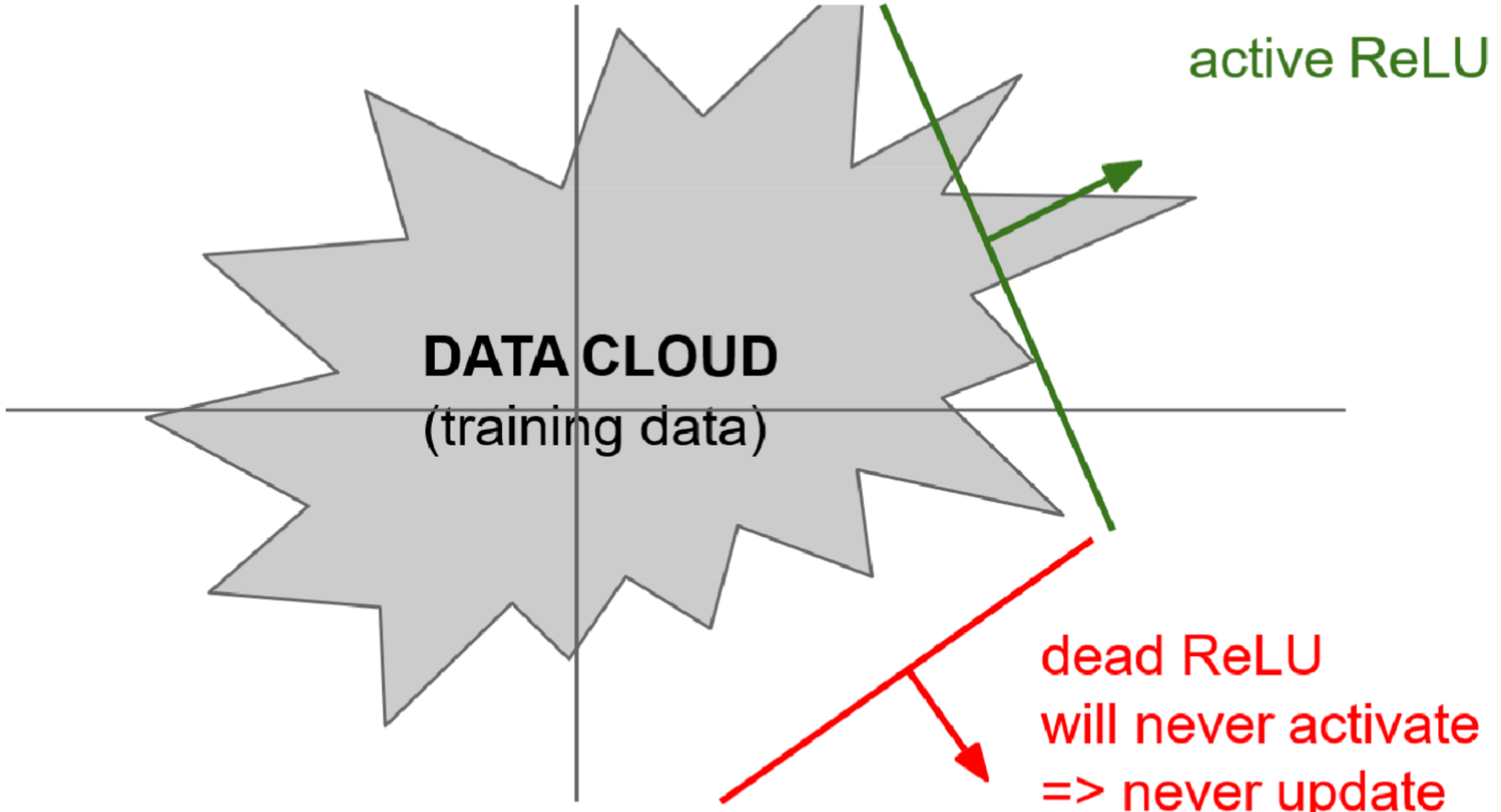
f(x) = max(0,x)



ReLU
(Rectified Linear Unit)

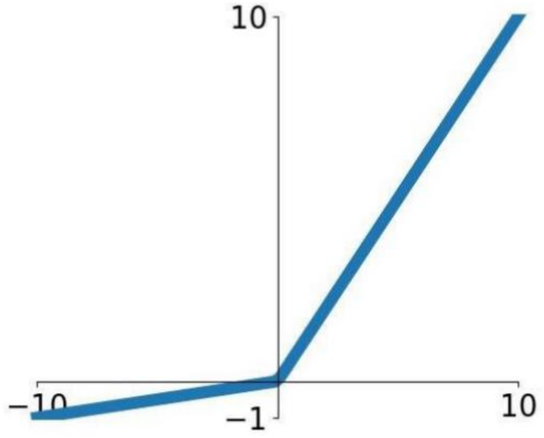
특징	- x>0 vanishing gradient 발생 X - Very computationally efficient - Fast convergence
문제	- x<0 Vanishing gradient - Not zero-centered

2.4 ReLU



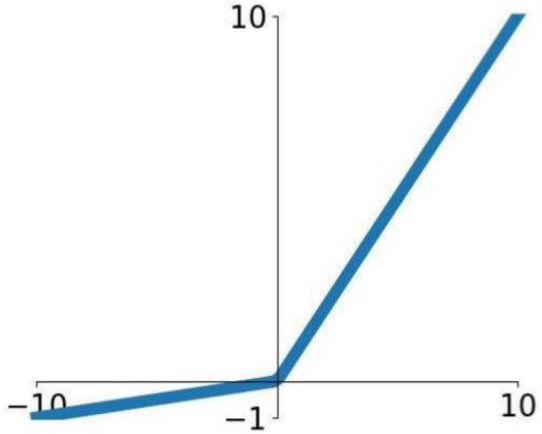
-> Dead ReLU를 방지하기 위해 bias 값을
아주 작은 양수(0.01)로 초기화 하기도 함

2.5 Leaky ReLU & PReLU& ELU



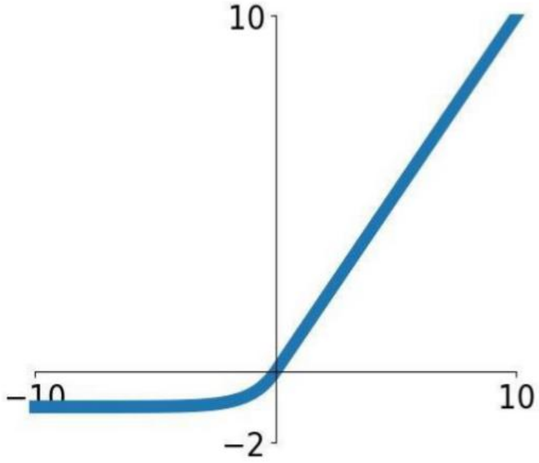
Leaky ReLU

$$\max(0.1x, x)$$



Parametric ReLU

$$\max(\alpha x, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

2.6 Maxout

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

특징	- Generalize ReLU and Leaky ReLU - Vanishing gradient 발생 X
문제	- Doubles the number of parameter/neuron

2.4 TLDR

1. 디폴트로 ReLU를 사용하세요
2. Leaky ReLU/Maxout/ELU 시도해 보세요
3. tanh는 사용하더라도 기대는 하지 마세요
4. 시그모이드는 더 이상 사용하지 마세요

03. Data preprocessing, Weight Initialization

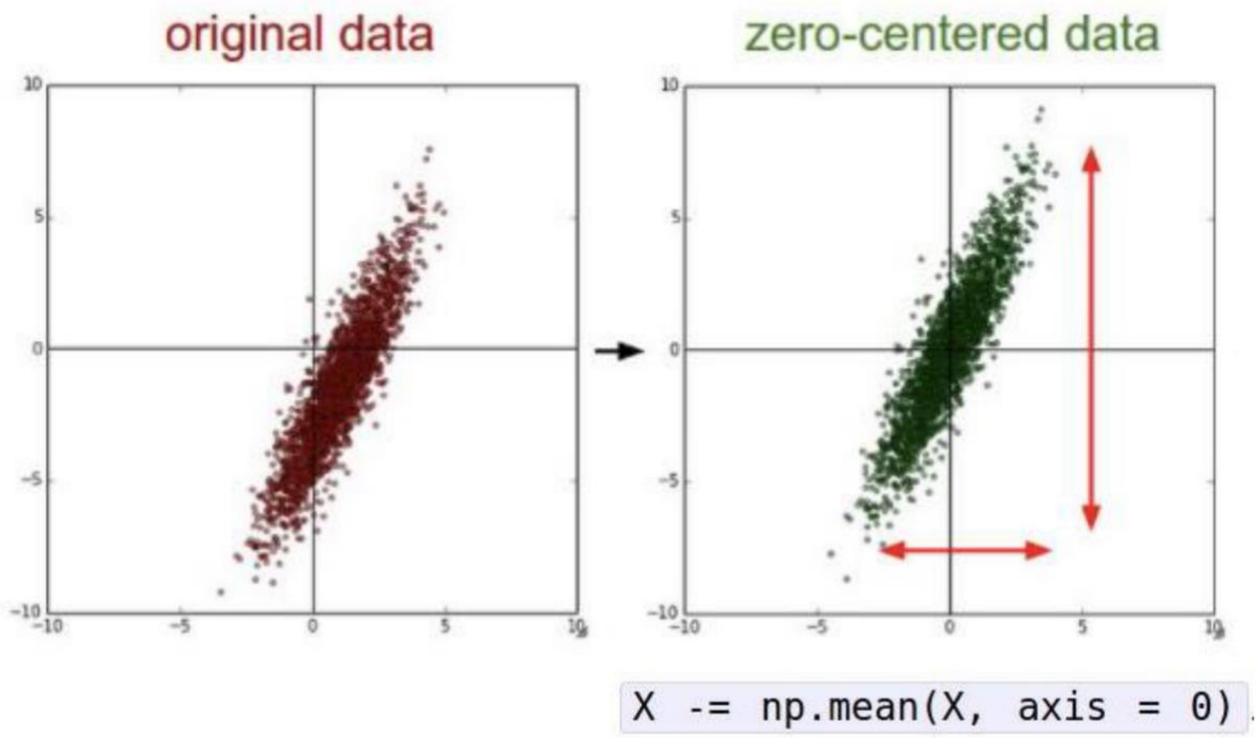
3.1 Data preprocessing

- 데이터 전처리의 필요성
- 데이터 전처리 방법
    1. 평균 차감(mean subtraction)
    2. 정규화(normalization)
    3. PCA와 Whitening

3.1 Data preprocessing

평균 차감 (Mean subtraction)

: 데이터의 모든 featrue값에 대하여 평균값을 차감하는 방법




```
X -= np.mean(X, axis = 0)
```

데이터 행렬 X는 D차원의 데이터 벡터 N개로 이루어진 NxD 행렬이라고 가정

3.1 Data preprocessing

정규화(normalization)

Scale: 어떤 특성이 가지고 있는 값의 범위

'두 특성의 스케일 차이가 크다'

	당도(1-10)	무게(500-1000)
사과1	4	540
사과2	8	700
사과3	2	480

3.1 Data preprocessing

정규화(normalization)

: 각 차원의 데이터가 동일한 범위 내의 값을 갖도록 하는 방법

Feature scaling ‘스케일을 조정한다’

1) Standardization(z-score normalization)

2) Min-Max normalization

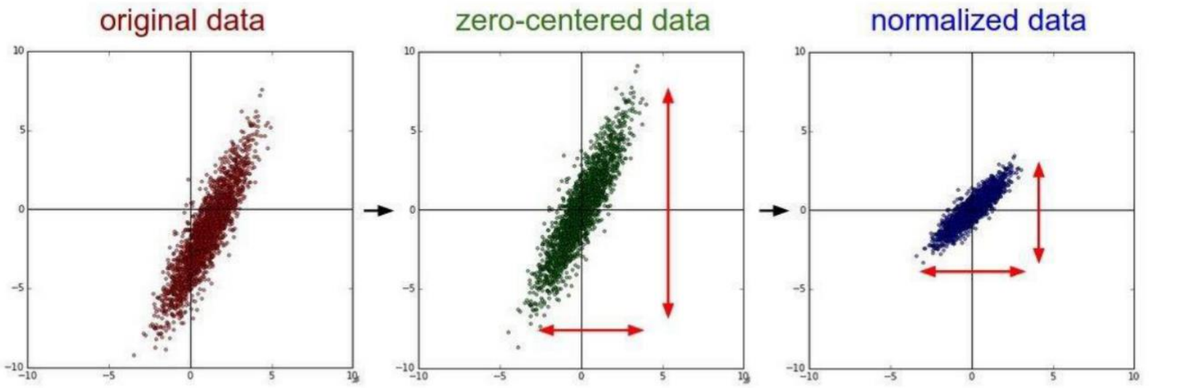
다만 이미지 데이터에서는 정규화가 필요하지 않다.

Zero-centering only!

3.1 Data preprocessing

정규화(normalization)

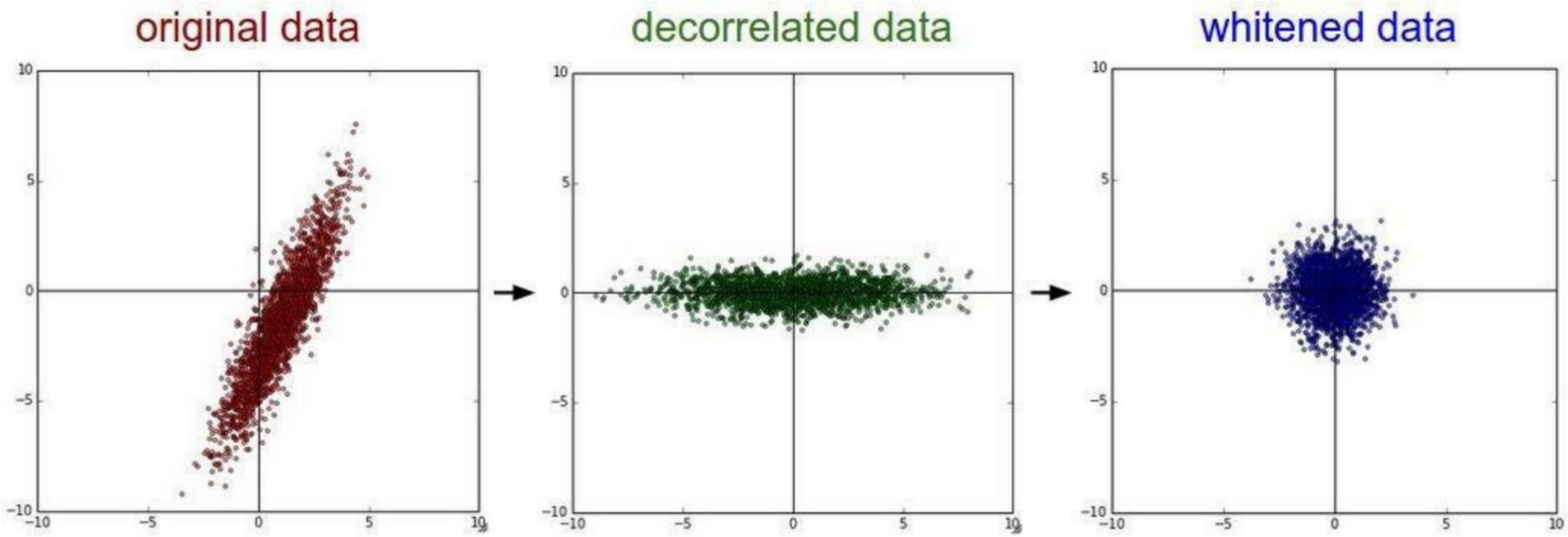
1) Standardization(z-score normalization $x_i\_new = \frac{x_i - mean(x)}{std(x)}$



2) Min-Max normalization $x_i\_new = \frac{x_i - min(x)}{max(x) - min(x)}$

3.1 Data preprocessing

PCA & Whitening



3.1 Data preprocessing

TLDR: In practice for Images: center only

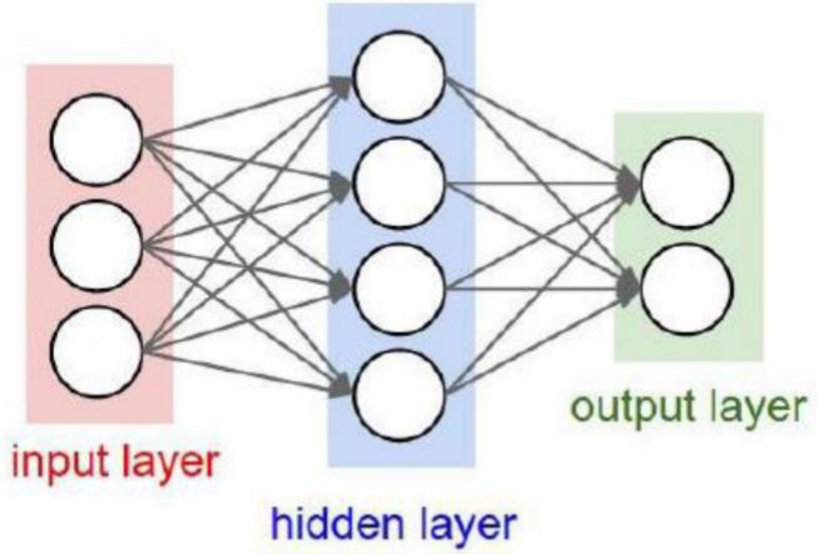
e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (e.g. VGGNet)
  (mean along each channel = 3 numbers)

Not common to normalize variance, to do PCA or whitening

3.2 Weight Initialization

- Q: what happens when W=0 init is used?



가중치가 0이라서 모든 뉴런은 모두 다 같은 연산을 할 것이고,
gradient 가 동일하게 될 것이다.

-> symmetry breaking이 일어나지 않음

3.2 Weight Initialization

0에 가까운 random number로 초기화

- 가중치 초기화의 기본적인 idea
- 정규분포 사용
- Weight decay: 가중치 매개변수의 값이 작아지도록 학습하는 방법

-> 오버피팅을 억제하는 테크닉

```
W = 0.01* np.random.randn(D,H)
```

3.2 Weight Initialization

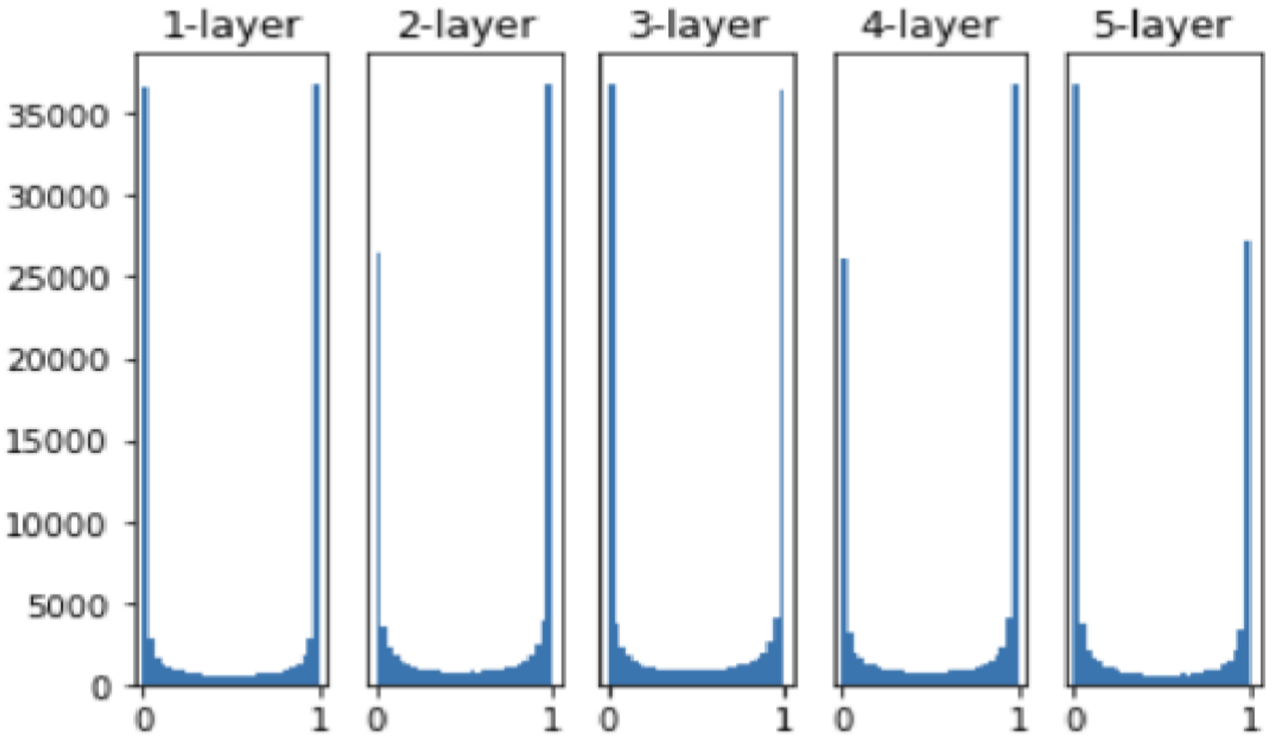
0에 가까운
random number로 초기화

-> But proplems
with deeper network!

```python
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

input_data = np.random.randn(1000, 100)  # 1000개의 데이터
node_num = 100  # 각 hiddenlayer의 뉴런(노드) 수
hidden_layer_size = 5  # hiddenlayer 5개
activations = {}  # 이곳에 활성화함수 결과값 저장

for i in range(hidden_layer_size):
    if i != 0:
        x = activations[i-1]
    w = np.random.randn(node_num, node_num) * 1
    # 표준편차가 1인 정규분포 이용
    a = np.dot(x, w)
    z = sigmoid(a)
    activations[i] = z

for i, a in activations.items(): # 히스토그램 그리기
    plt.subplot(1, len(activations), i+1)
    plt.title(str(i+1) + "-layer")
    if i != 0: plt.yticks([], [])
    plt.hist(a.flatten(), 30, range=(0,1))
plt.show()
```

표준편차가 1인 정규분포

3.2 Weight Initialization

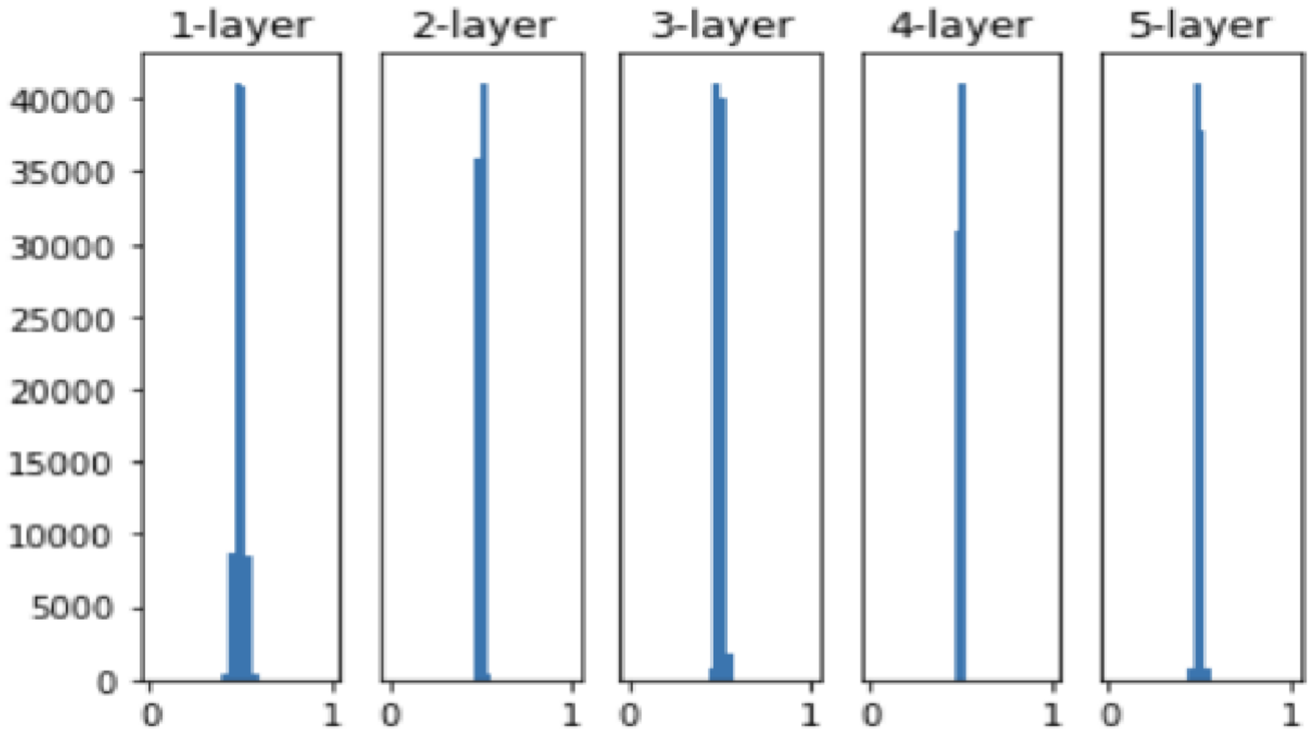


Vanishing gradient 발생

3.2 Weight Initialization

```
# w = np.random.randn(node_num, node_num) * 1
w = np.random.randn(node_num, node_num) * 0.01
```

표준편차가 0.01인 정규분포

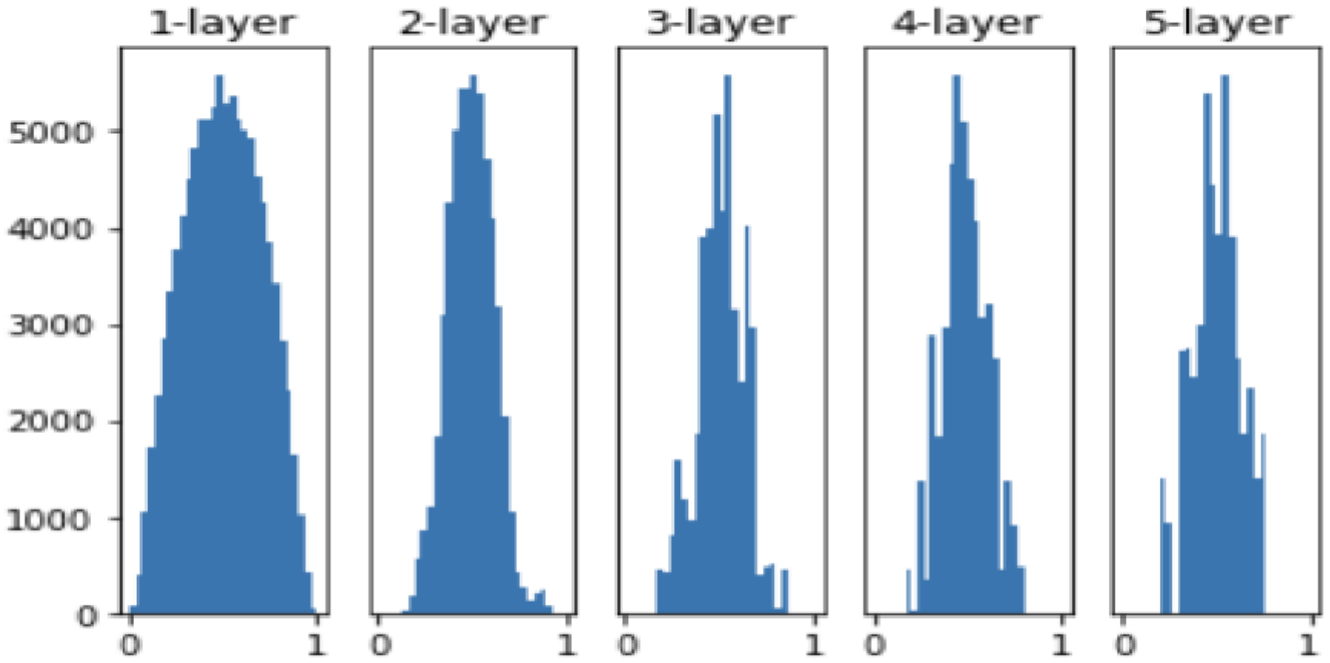


3.2 Weight Initialization

Xavier initialization

```
node_num = 100 # 앞 층의 노드 수
w = np.random.randn(node_num, node_num) / np.sqrt(node_num)
```

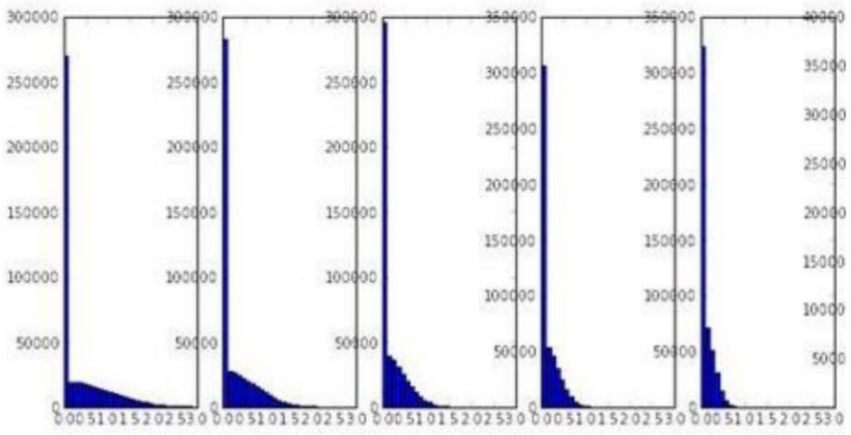
node_in, node_out　　node_in



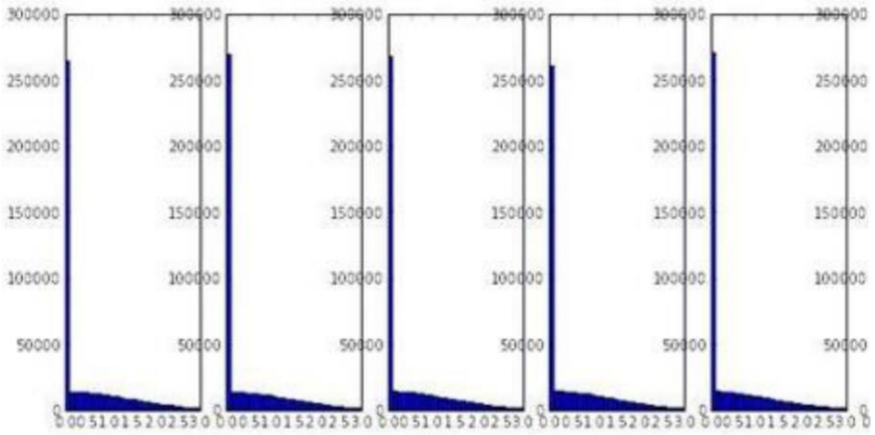
3.2 Weight Initialization

He initialization

```
#w = np.random.randn(node_in, node_out) / np.sqrt(node_in)
w = np.random.randn(node_in, node_out) / np.sqrt(node_in/2)
```



Xavier 초깃값



He 초깃값

04. Batch Normalization

4.1 Batch Normalization

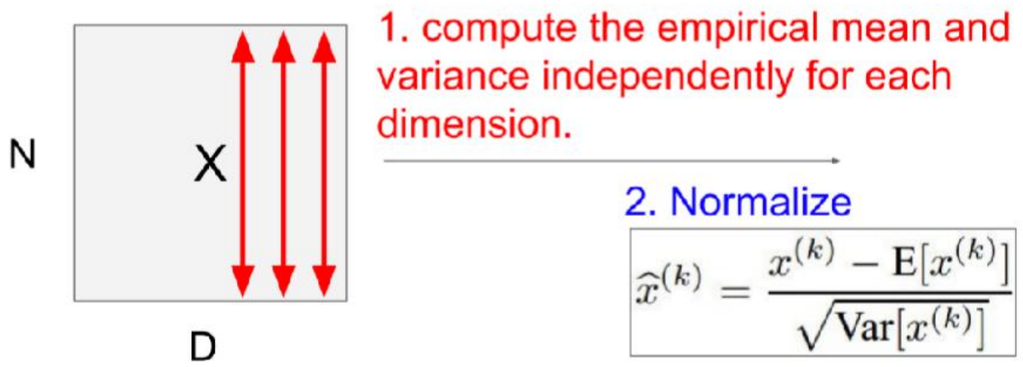
활성화 값 분포가 적당히 퍼져 있는 것은 학습이
원활하게 수행되어 있도록 돕는다.

Batch Normalization 이용하여 활성화 값
분포를 적당히 퍼트리도록 강제할 수 있다

4.1 Batch Normalization

- Batch 별로 평균과 분산을 각각 구해 정규화
- 레이어의 입력이 unit gaussiam이 되도록 강제하는 것 -> saturation방지
- Gradient Vanishing 이 일어나지 않도록 하는 방법 중 하나

- Activation function의 변화, Careful Initialization, Small learning rate 등으로 해결했지만, 이런 간접적인 방법보다 training하는 과정 자체를 안정화해서 학습 속도를 가속시킬 근본적인 방법임



1) N: Batch 당 N개의 학습 데이터
   D: 데이터의 차원

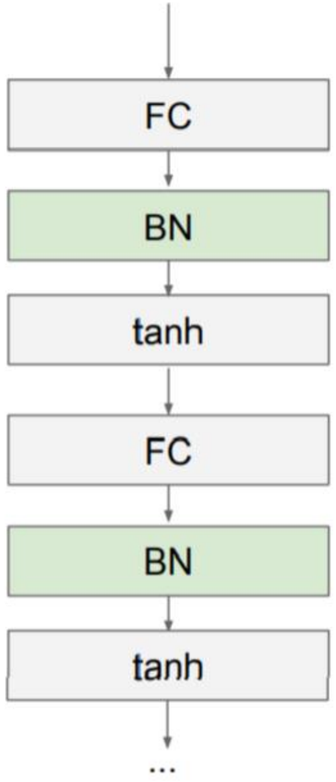
2) 각 차원별(feature element 별로) 평균을 각각 구함

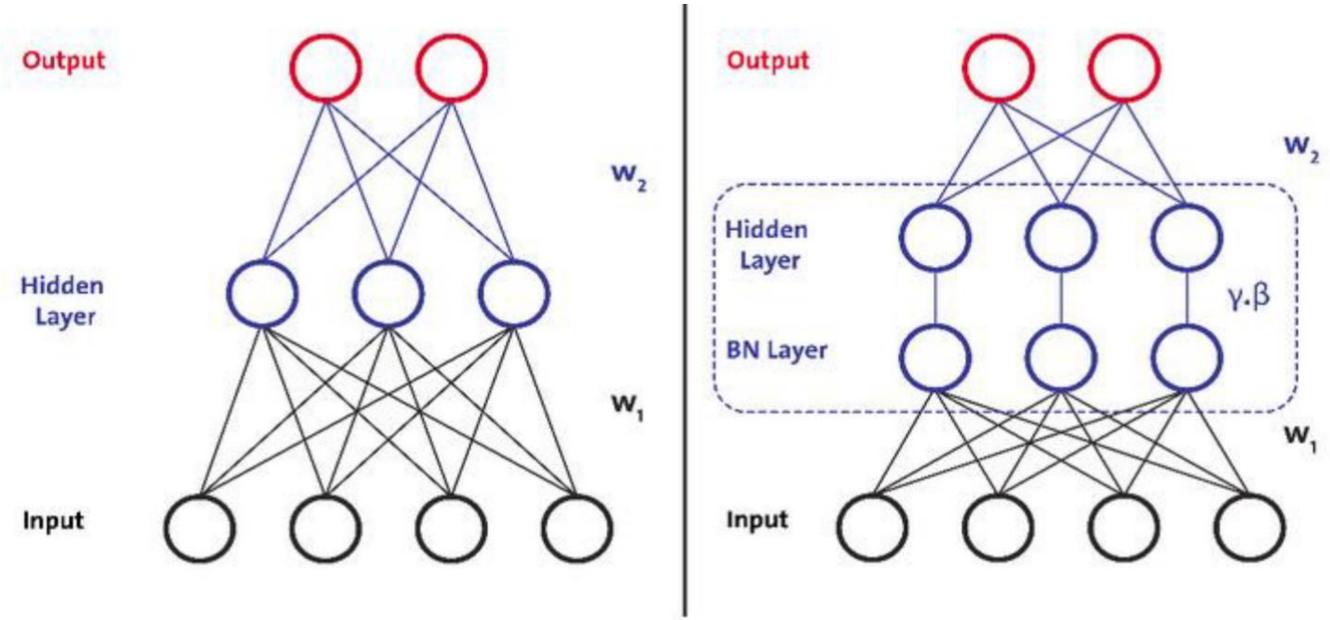
3) Batch 내에 이걸 전부 계산해 normalize

☞ 정규화 결과, 평균은 0, 분산은 1이 됨

4.1 Batch Normalization

- 연산은 FC나 Conv layer 직후에 넣어줌
- 깊은 네트워크에서 각 레이어의 w가 지속적으로 곱해져 발생한 bad scaling effect를 normalization은 상쇄시킴





활성화 함수의 앞/뒤에 BN Layer을 삽입하여
데이터의 편향을 줄일 수 있다.

4.1 Batch Normalization

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

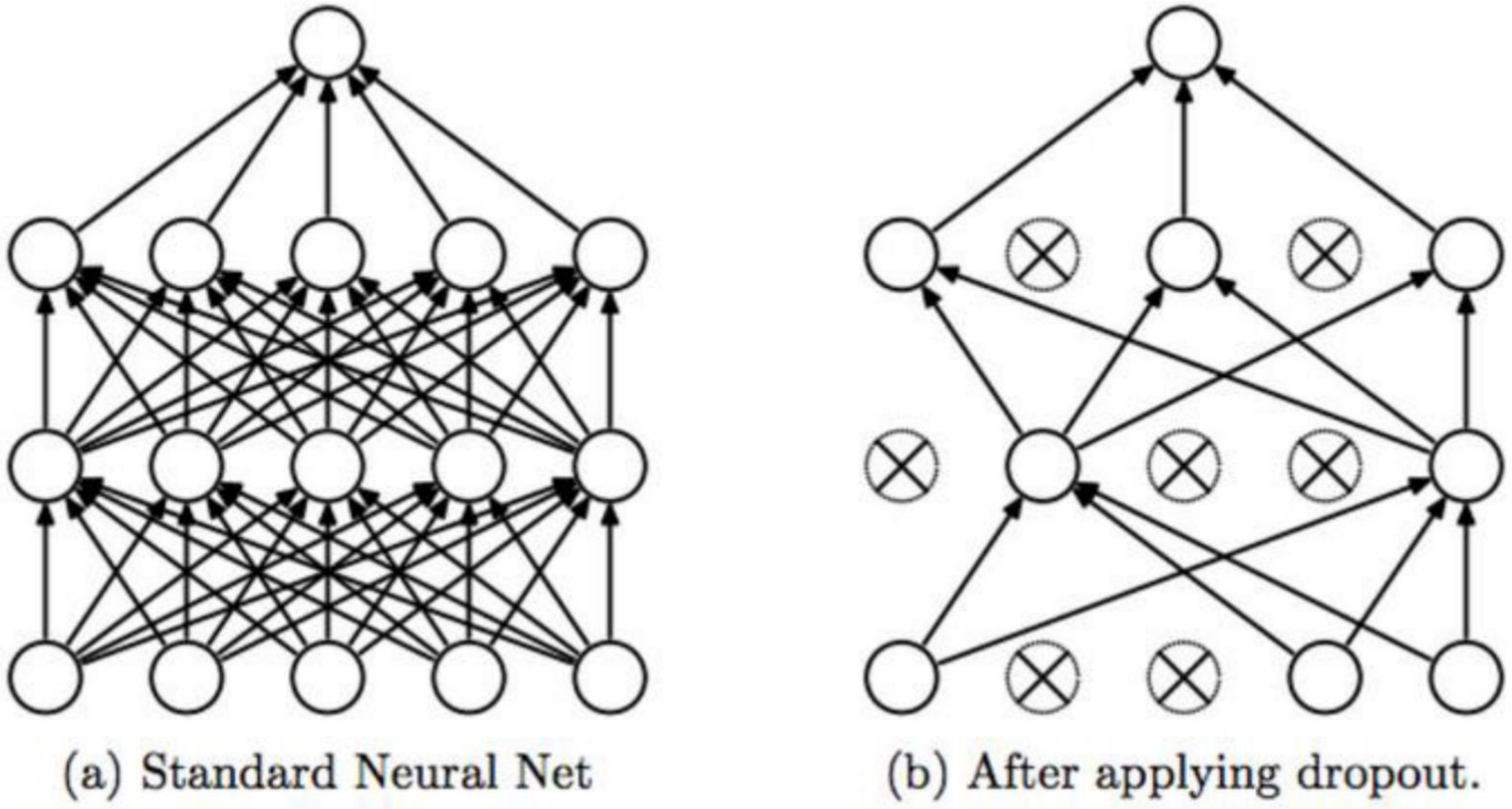
$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

- BN은 입력값을 not saturated 한 영역에 있게 만든다.
- 하지만, 무조건 saturation을 막기 보단, saturation의 조절을 학습할 수 있다면 더 효율적인 결과를 얻음
  - ✓ 감마는 scaling, 베타는 shift의 효과
  - ✓ 감마값과 베타값을 학습을 통해 찾음
- 감마값에 분산값을, 베타값에 평균값을 넣으면 unit gaussian 이전의 원래 상태에 비슷하게 복구 가능

4.1 Batch Normalization

Dropout: Prevent Overfitting



(a) Standard Neural Net

(b) After applying dropout.

4.1 Batch Normalization

Dropout: Prevent Overfitting

Hidden Layer의 Neuron 임의 누락

→ 하나의 특정 특징에 대한 편향 방지,
다양한 특징이 골고루 이용될 수 있도록 함

→ Overfitting 방지

4.1 Batch Normalization

Dropout: Prevent Overfitting

Forward pass 과정에서 일부 뉴런의 값을 임의로 0으로 만들어 구현

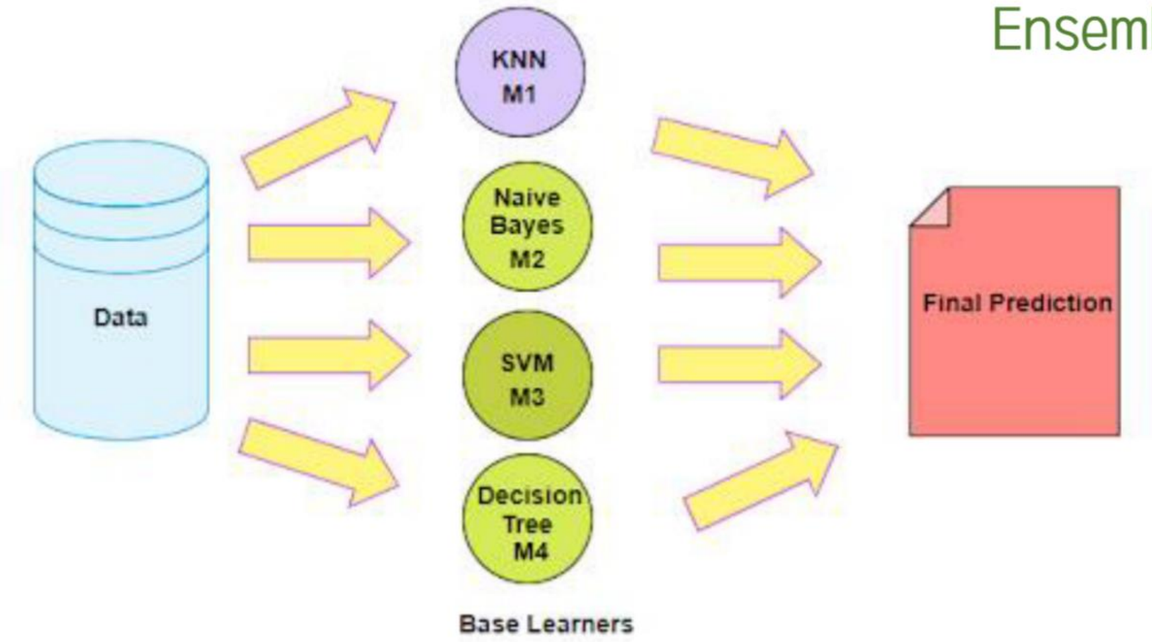
- self.mask에 삭제할 뉴런을 False라고 표시
  - X와 형상이 같은 배열을 무작위 생성, dropout_ratio보다 큰 원소만 True로 설정

```python
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg = True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout*self.mask
```

4.1 Batch Normalization

Dropout: Prevent Overfitting

시행할 때마다 누락되는 Neuron이 바뀜
Sub Network의 다양성 ↑
Ensemble 효과와 유사



05. Babysitting the Learning process

5.1 Babysitting the Learning process

<트레이닝을 모니터링 하는 방법>

1. 데이터 전처리
2. 히든 레이어나 개수나 뉴런의 개수 등 기본 구조에 관한 아키텍쳐 정하기
3. Loss가 잘 나오는지 확인

Double check that the loss is reasonable:

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 0.0)
print loss
```

disable regularization

```
2.30261216167
```

loss ~2.3.
"correct " for
10 classes

returns the loss and the
gradient for all parameters

```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e3)
print loss
```

crank up regularization

```
3.06859716482
```

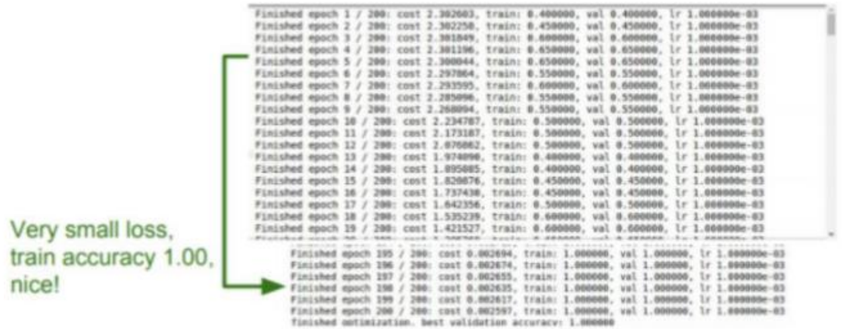
loss went up, good. (sanity check)

- Softmax 함수, 규제값을 0으로 설정 => -log1/c 값으로 2.3
- 규제값 올렸을 때, loss가 어떻게 변하는지 sanity check

5.1 Babysitting the Learning process

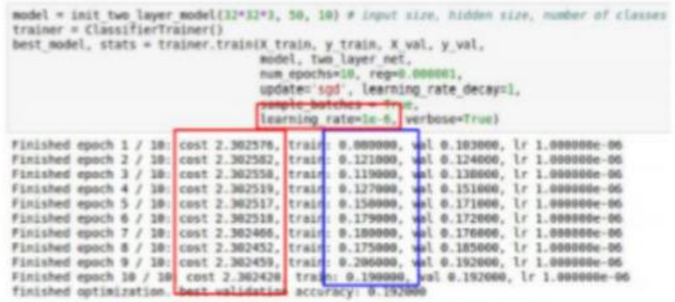
4. 실제로 훈련

Lets try to train now…



- 매우 적은 데이터셋을 넣었을 때, 과적합 되면 모델이 정상적으로 작동중임

5. 규제값과 학습률 찾기 => 결과 확인 반복

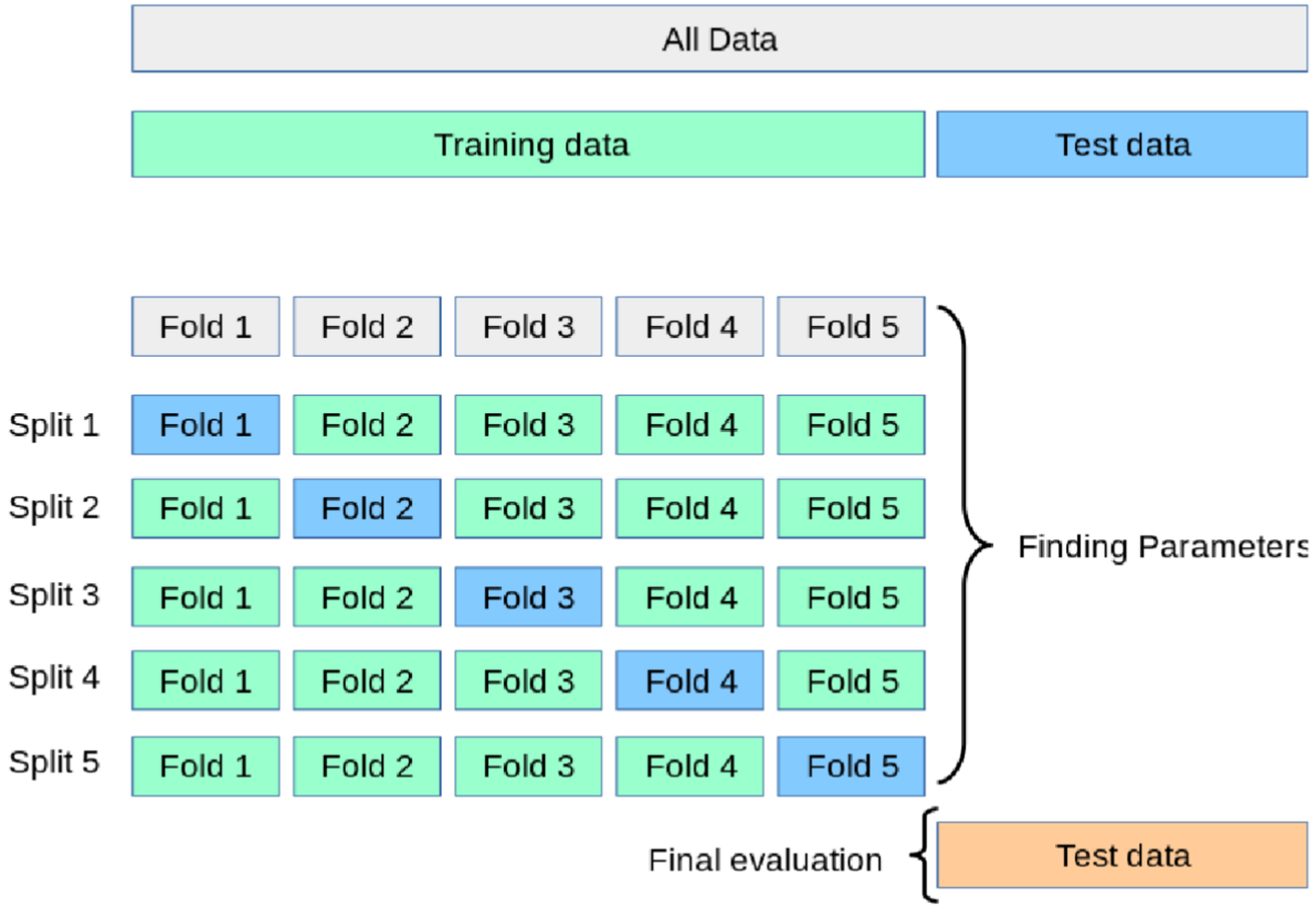


- 작은 regularization값을 넣고, 학습률을 낮게/높게 설정해보고 결과 확인 반복

=> 반복으로 적절한 학습률을 찾아감

06. Hyperparameter Optimization

6.1 Hyperparameter Optimization



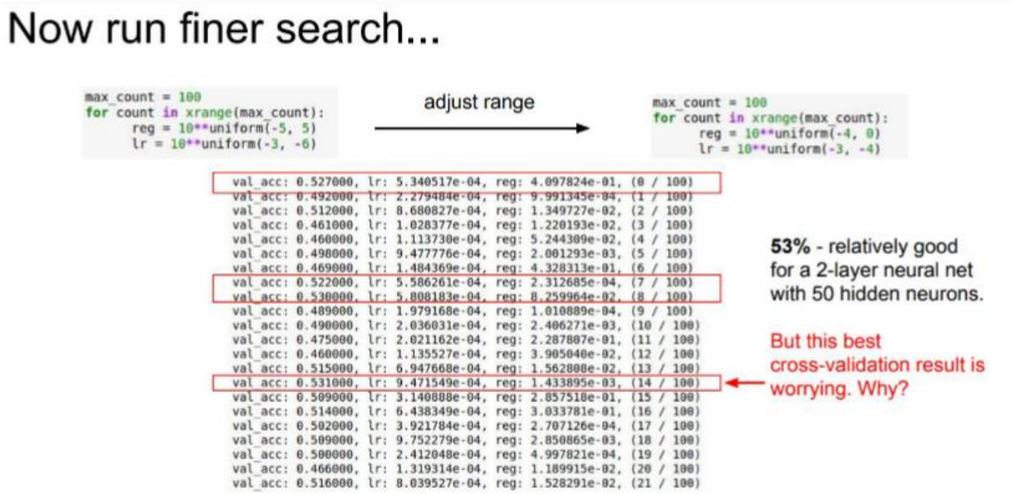
Cross-Validation Strategy

Train at your Train Set, Validate at your Validation Set

6.1 Hyperparameter Optimization

< 하이퍼 파라미터 최적화 >

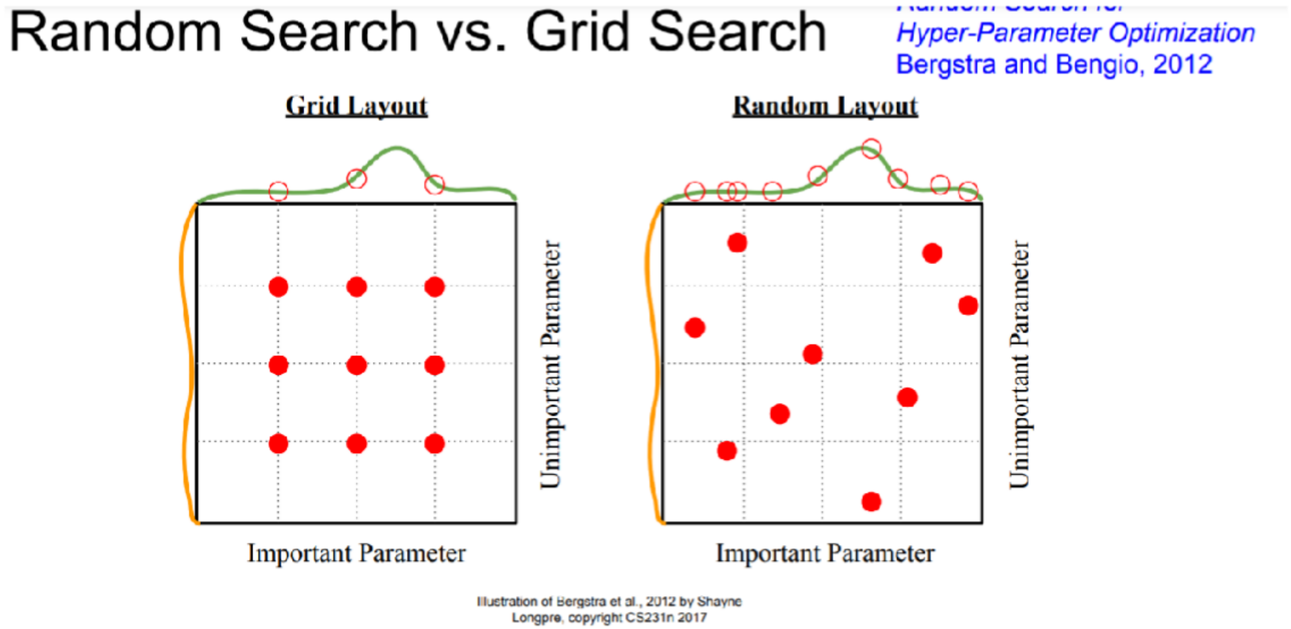
1. Hyperparameter 값을 우선 설정
2. 범위 내에서 파라미터 값을 무작위로 추출(Random search)
3. Validation set을 이용하여 평가하는 Cross-validation 진행
4. 여러 번 반복 중 정확도를 체크하면 hyperparameter값의 범위를 좁힘



- Hyperparameter값을 일일이 반복을 통해 찾는 과정 보단 search 방법들을 사용하는 것이 더욱 효과적이다.

Few epochs
→ Longer Running Time & Finer Search

6.1 Hyperparameter Optimization



Grid Search

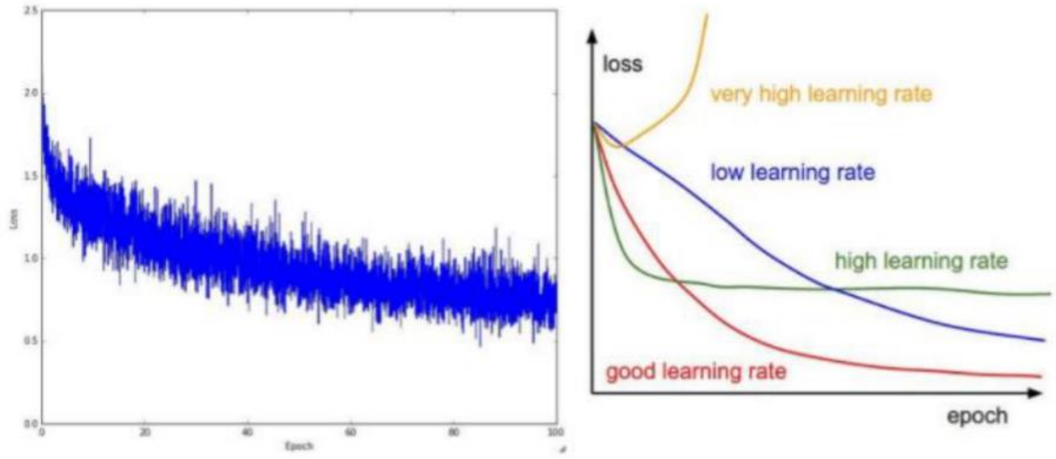
하이퍼파라미터 값의 범위를 지정해 일정한 간격을 두고 하이퍼파라미터 값을 지정, 각 값들에 대해 측정한 성능을 비교하여 가장 높은 성능을 보인 값 채택

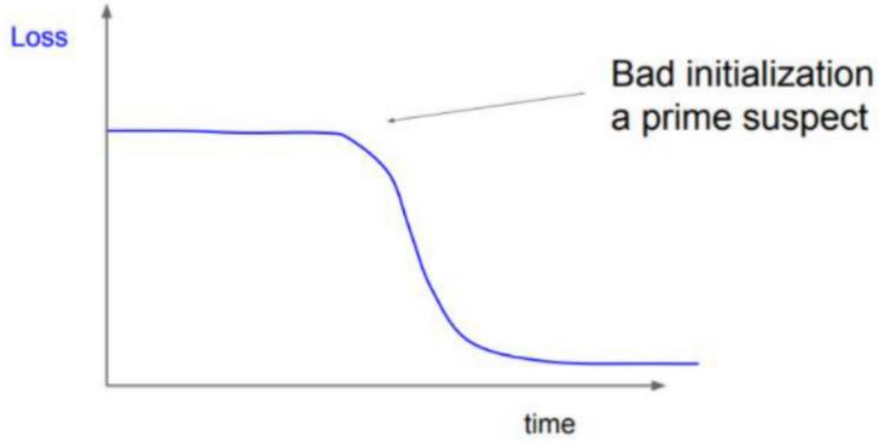
Random Search

하이퍼파라미터 값의 범위를 지정해 범위내의 하이퍼파라미터 값들을 랜덤 샘플링하여 값을 지정, 불필요한 연산 수행 줄여 더 빠르게 찾을 수 있음. 더 많이 사용됨.

6.1 Hyperparameter Optimization

Monitor and visualize the loss curve





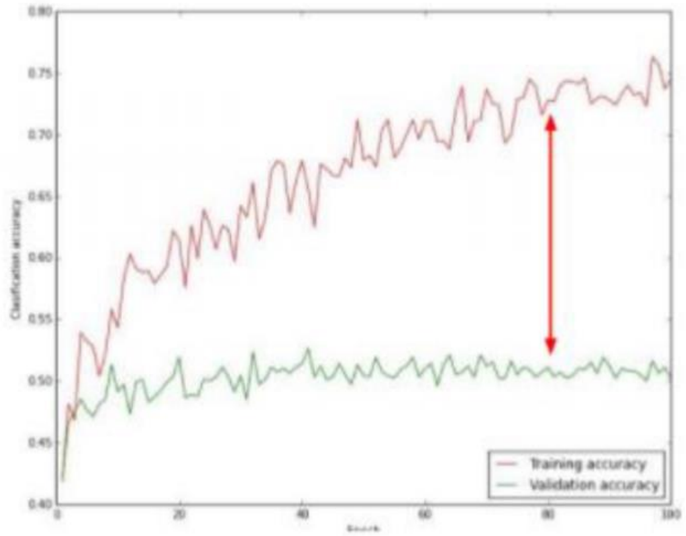
Loss Curve

모니터링시 하이퍼 파라미터가 적합한지 아닌지 평가
학습률의 경우, 빨간색 선이 가장 좋은 형태

초기에 평평한 모양의 로스 커브가 나올 시 , 초기화가 잘못되었을 가능성이 크다.

6.1 Hyperparameter Optimization

Monitor and visualize the accuracy:



big gap = overfitting
=> increase regularization strength?

no gap
=> increase model capacity?

- 트레이닝 accuracy와 검증 accuracy의 갭이 클 경우 과적합 상태이니 규제값 강도를 올려봐야 한다.
- 반대로 갭이 없을 땐, model capacity를 늘려야 한다.

0.0 Summary

Summary

TLDRs

We looked in detail at:

- Activation Functions (use ReLU)
- Data Preprocessing (images: subtract mean)
- Weight Initialization (use Xavier init)
- Batch Normalization (use)
- Babysitting the Learning process
- Hyperparameter Optimization
  (random sample hyperparams, in log space when appropriate)