[6주차] Training neural networks Part 1

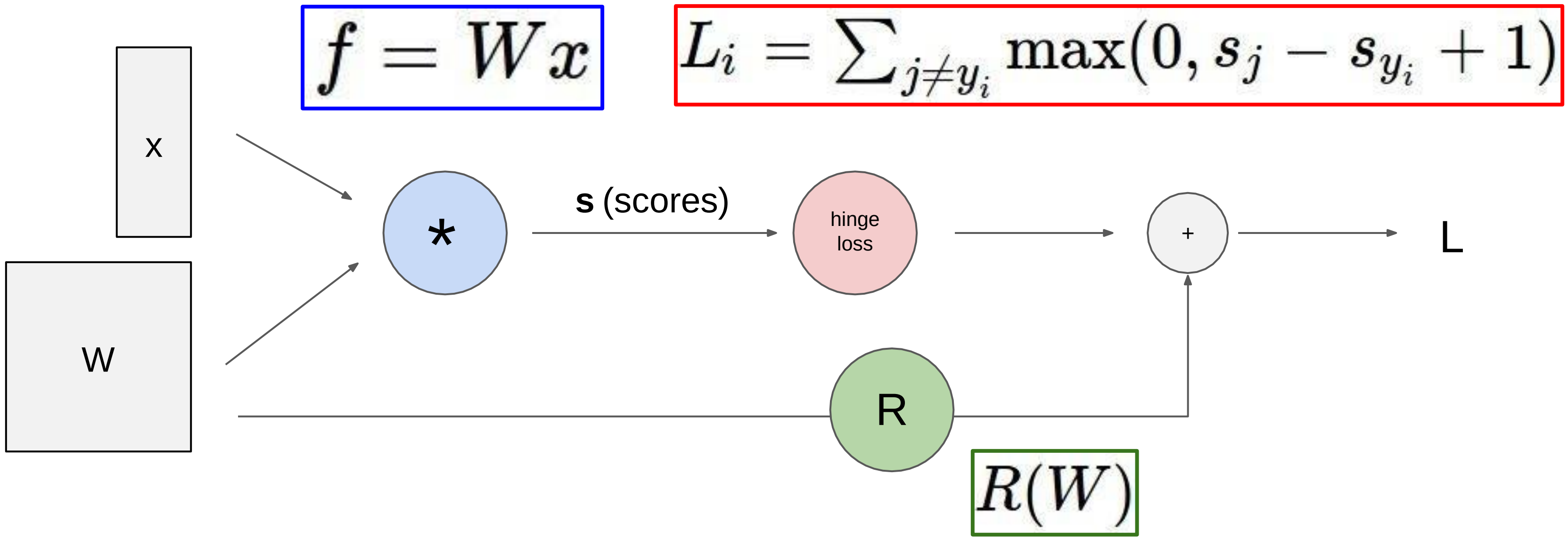
1기 구미진
1기 장예서

EURON

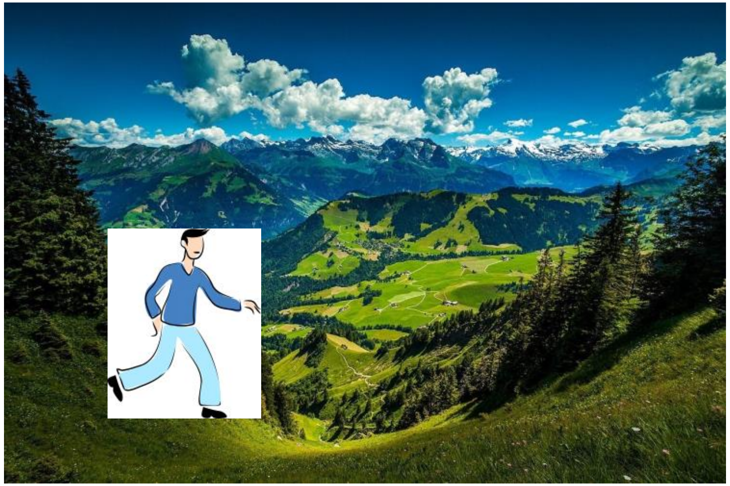
목차

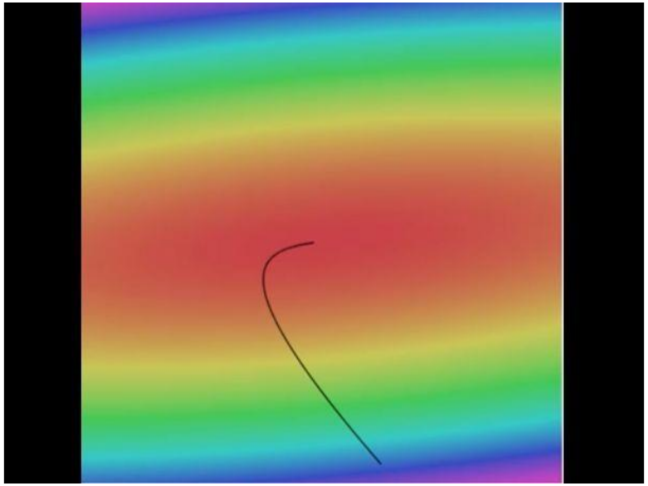
Review

Computational graphs



Learning network parameters through optimization





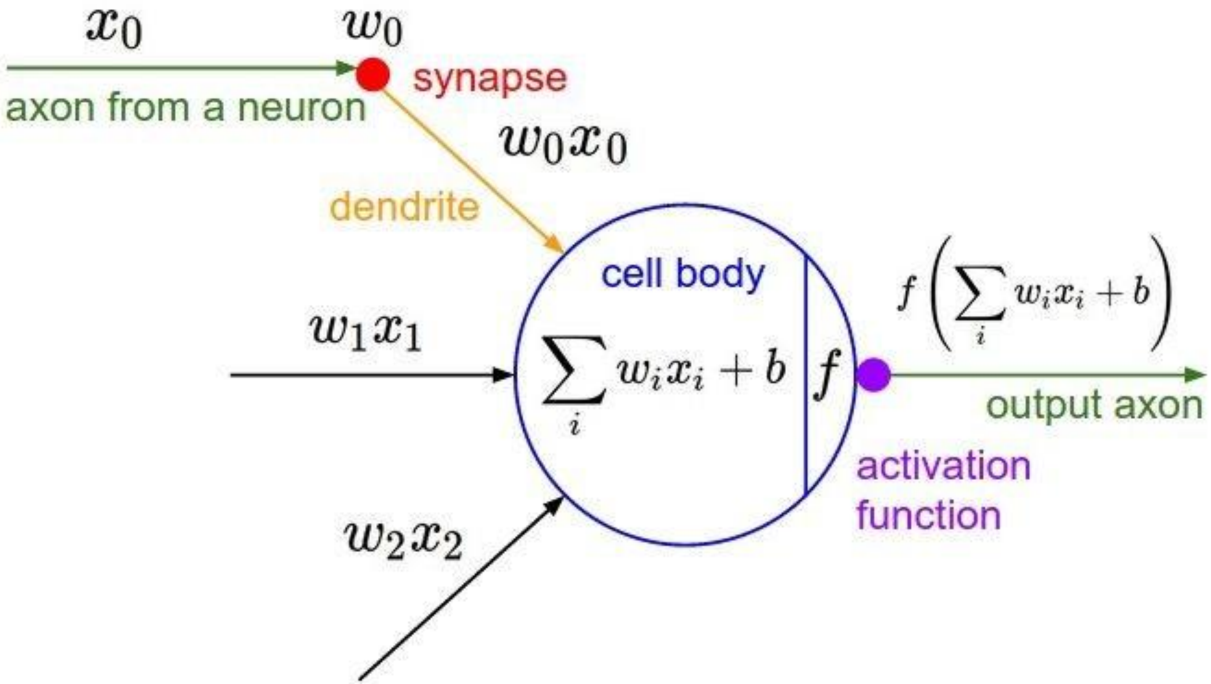
```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Mini-batch SGD

Loop:

1. **Sample** a batch of data
2. **Forward** prop it through the graph (network), get loss
3. **Backprop** to calculate the gradients
4. **Update** the parameters using the gradient

Activation function

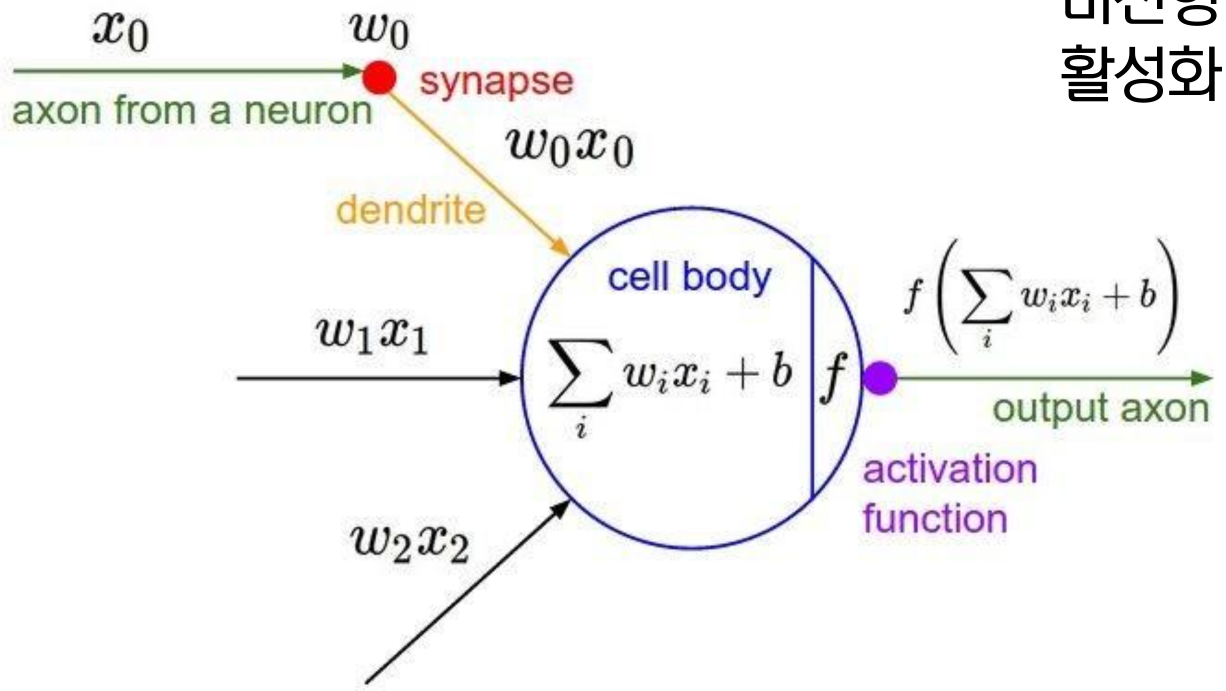


Activation function

과거 Neural Network의 문제점

1. GPU 성능
2. Underfitting -> 활성화 함수
3. Overfitting

Activation function

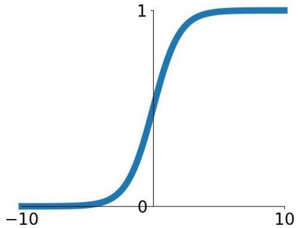


비선형 함수를
활성화 함수로 사용

Activation function의 종류

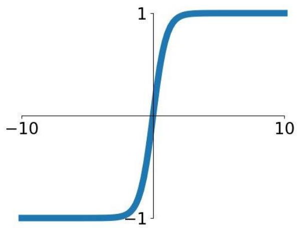
Sigmoid

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



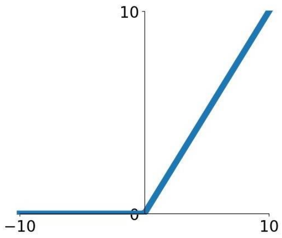
tanh

$$\tanh(x)$$



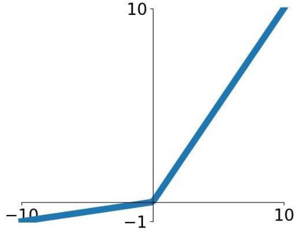
ReLU

$$\max(0, x)$$



Leaky ReLU

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

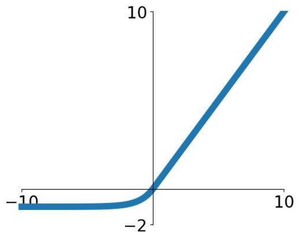


Maxout

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

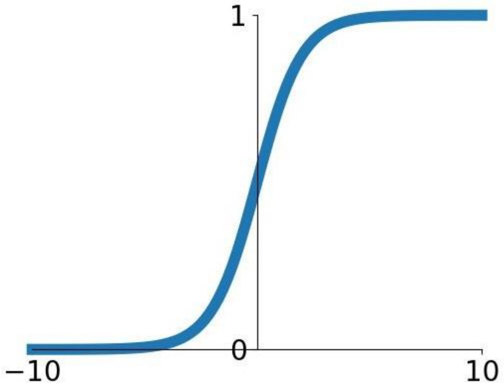
ELU

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



Sigmoid 함수

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

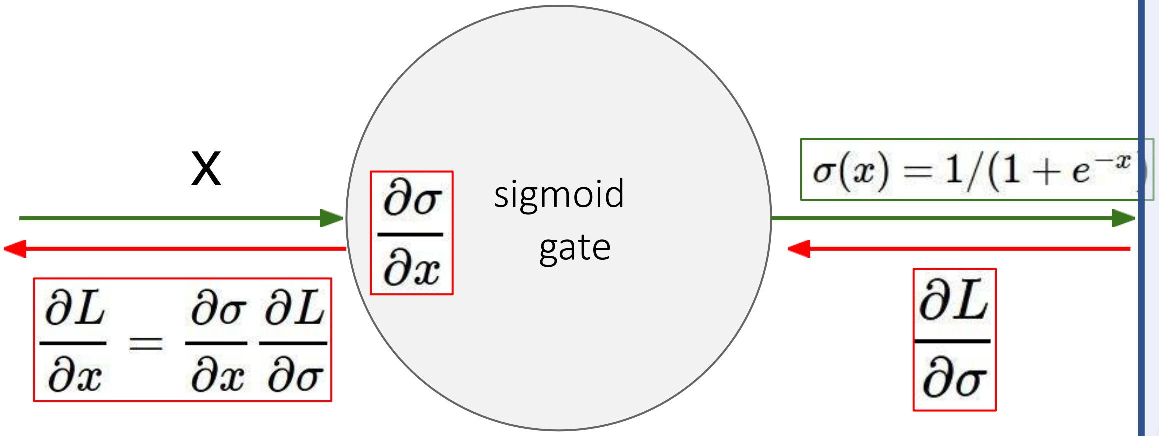


Sigmoid

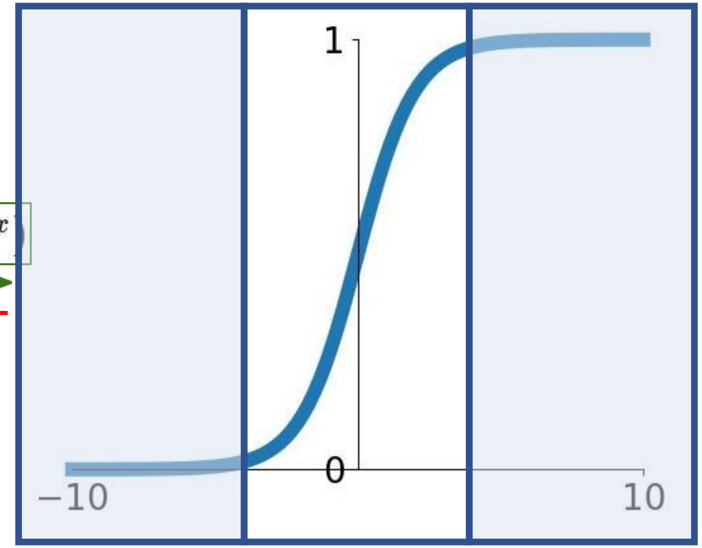
특징	- Squash function - Logistic function - Historically popular since they have nice interpretation as a saturating "firing rate" of neuron
문제	- exp() -> expensive computation - Vanishing gradient - Not zero-centered

Sigmoid 함수

문제2 vanishing gradient



Saturated
regime

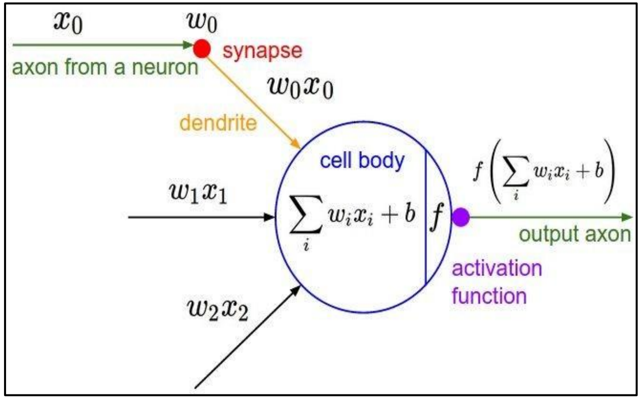


Active region
of sigmoid

Sigmoid 함수

문제3 Non zero-centered -> Slow convergence 초래

x(input)가 항상 양수인 경우를 생각해보면…



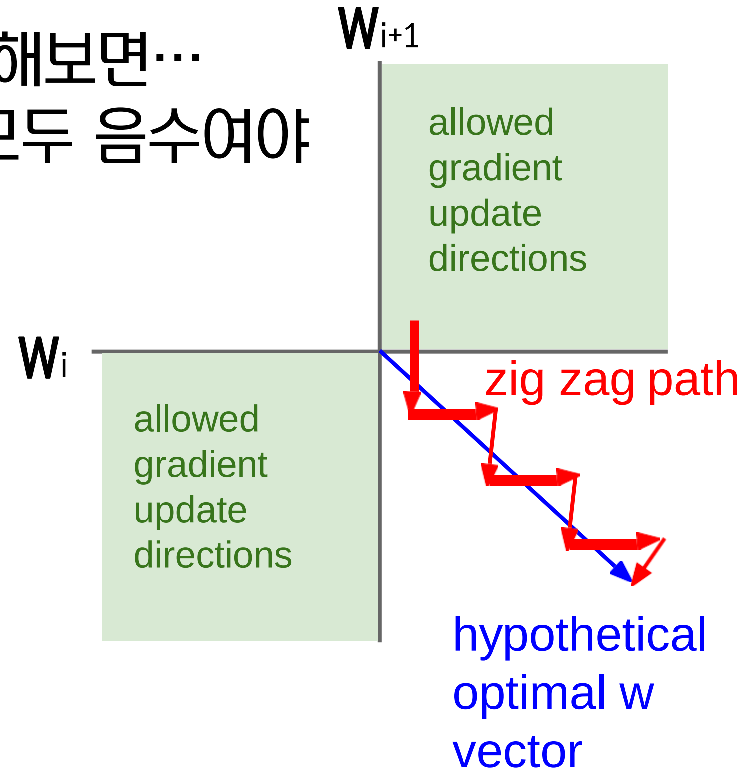
$$f\left(\sum_i w_i x_i + b\right) \qquad \frac{\partial f}{\partial w} = x$$

Sigmoid 함수

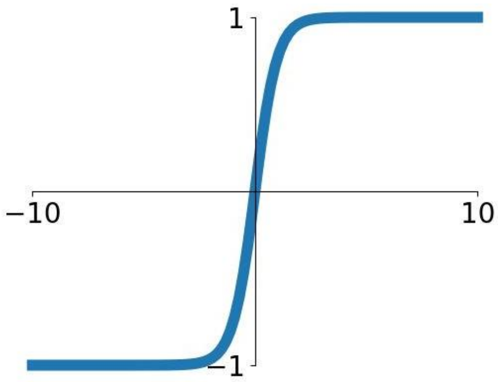
문제3 Non zero-centered

x(input)가 항상 양수인 경우를 생각해보면…
→ w의 gradient가 모두 양수거나 모두 음수여야
→ slow convergence를 초래

$$\boxed{\frac{\partial L}{\partial w}} = \frac{\partial L}{\partial f} \times \frac{\partial f}{\partial w} = \boxed{\frac{\partial L}{\partial f}} \times x$$



tanh 함수



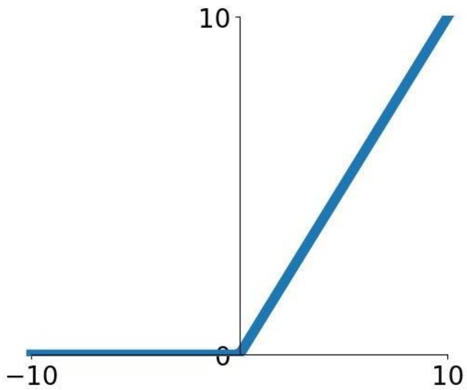
tanh(x)

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

ReLU 함수

Sigmoid 함수의 단점

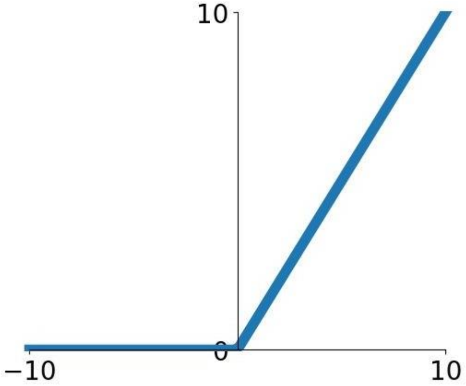
1. Vanishing gradient
2. Slow convergence



ReLU
(Rectified Linear Unit)

ReLU 함수

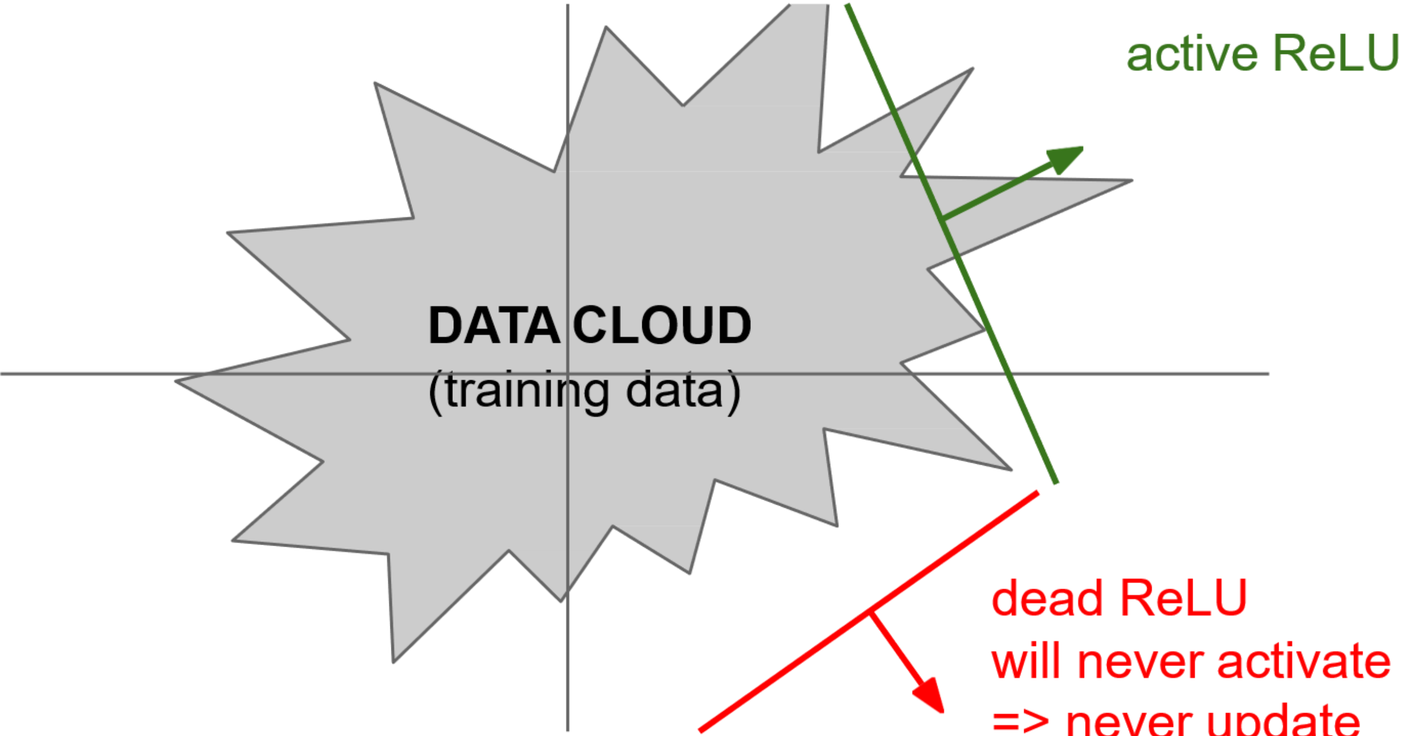
$f(x) = \max(0,x)$



ReLU
(Rectified Linear Unit)

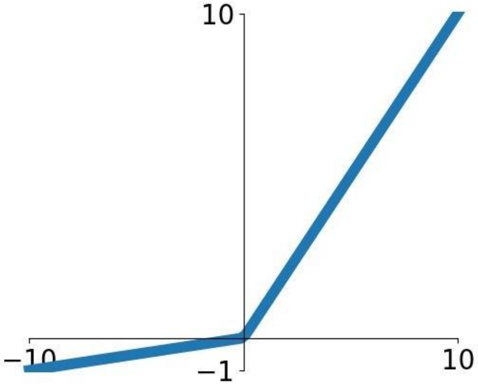
특징	- x>0 vanishing gradient 발생 X - Very computationally efficient - Fast convergence
문제	- x<0 Vanishing gradient - Not zero-centered

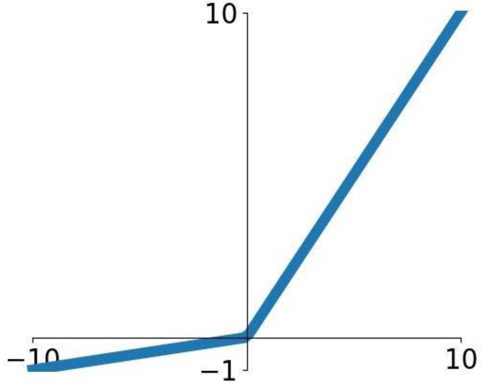
ReLU 함수

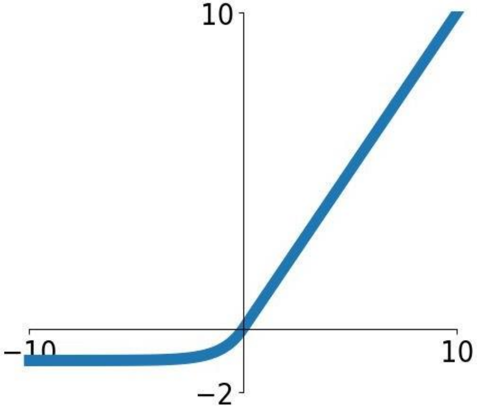


-> Dead ReLU를 방지하기 위해 bias 값을 아주 작은 양수(0.01)로 초기화 하기도 함

Leaky ReLU & PReLU & ELU







Leaky ReLU

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

Parametric ReLU

$$\max(\alpha \mathrm{x}, \mathrm{x})$$

ELU

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

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

실전에서는

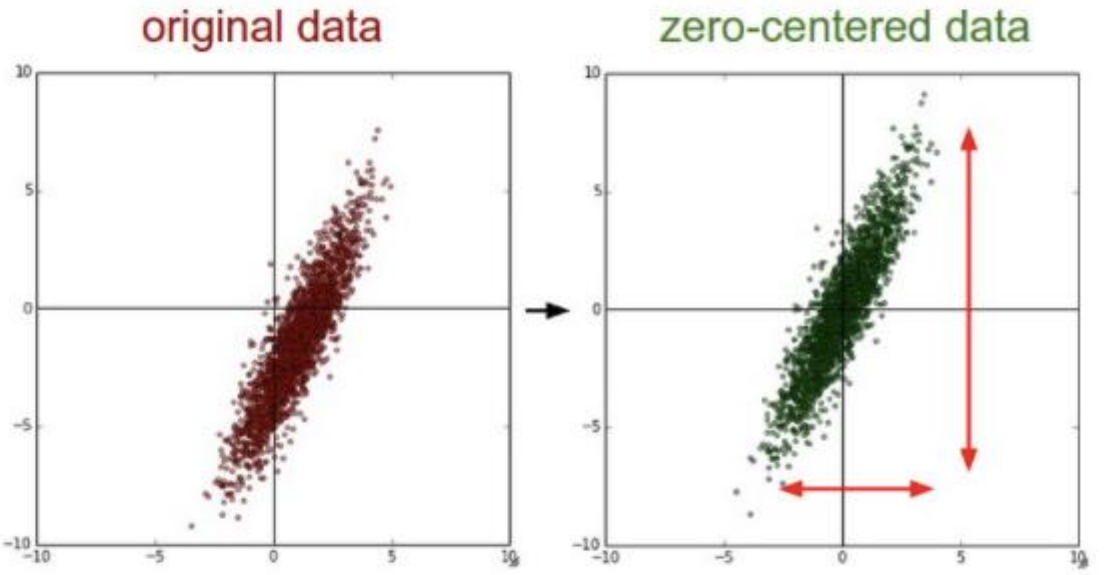
1. 디폴트로 ReLU를 사용하세요
2. Leaky ReLU/Maxout/ELU 시도해 보세요
3. tanh는 사용하더라도 기대는 하지 마세요
4. 시그모이드는 더 이상 사용하지 마세요

Data preprocessing

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

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

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

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

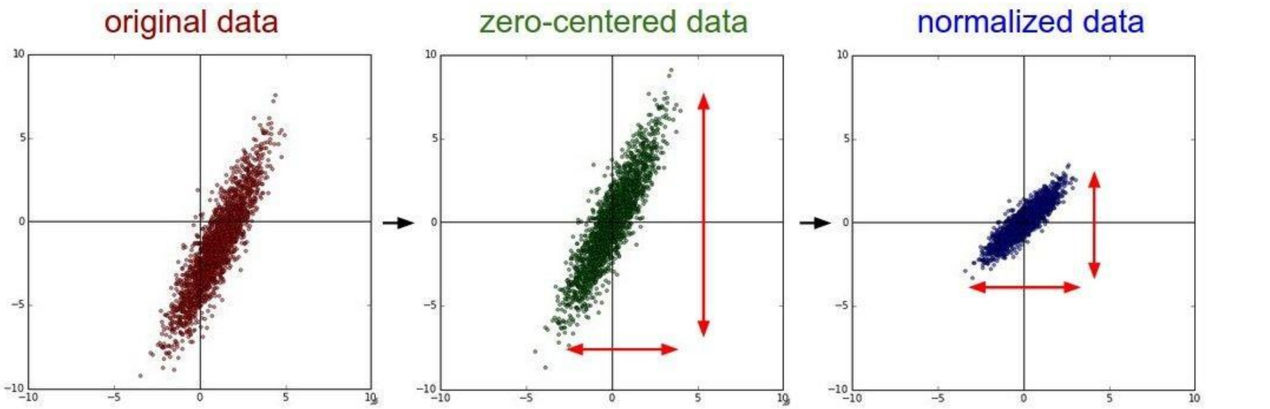
다만 이미지 데이터에서는 정규화가 필요하지 않다.

Zero-centering only!

정규화(normalization)

1) Standardization(z-score normalization

$$x_i\_new = \frac{x_i - mean(x)}{std(x)}$$



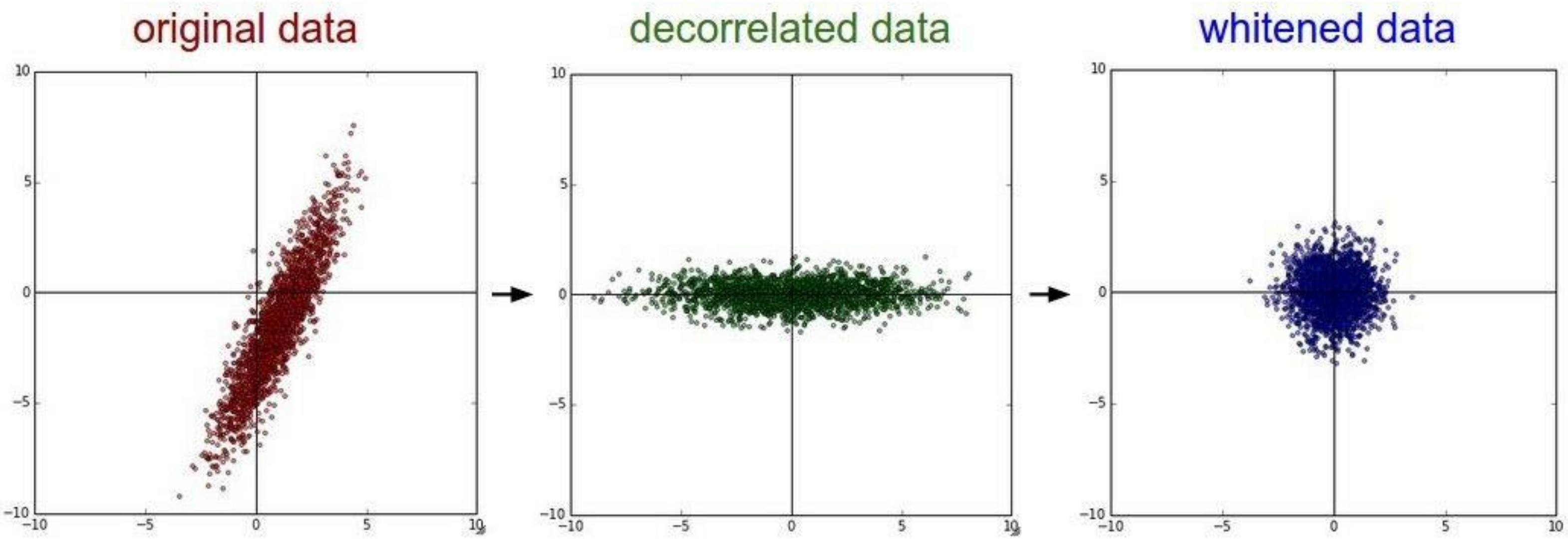

```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

2) Min-Max normalization

$$x_i\_new = \frac{x_i - min(x)}{max(x) - min(x)}$$

PCA & Whitening



(data has diagonal covariance matrix)

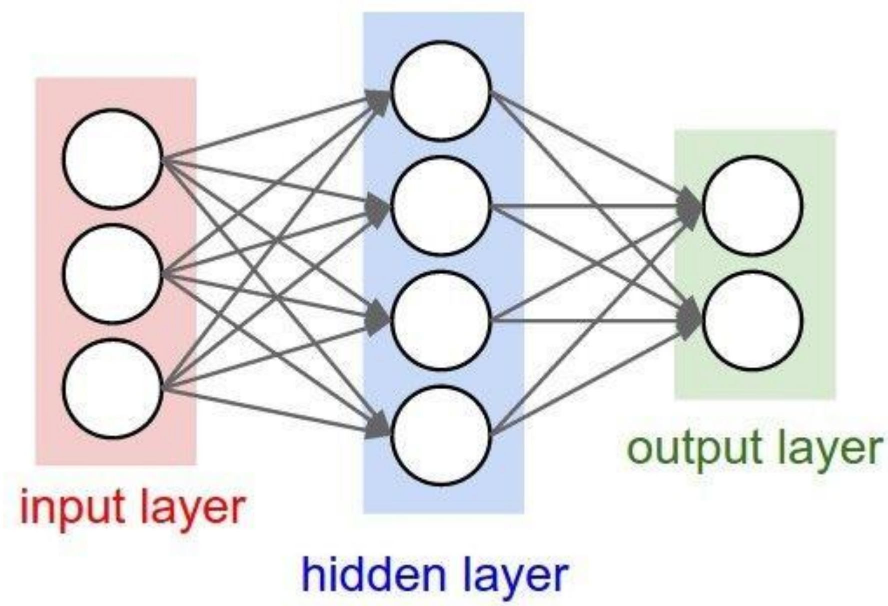
(covariance matrix is the identity matrix)

Weight initialization

Q: 가중치의 초깃값을 모두 0으로 설정한다면?

-> 모든 뉴런들이 동일한 연산을 수행함

-> symmetry breaking이 일어나지 않음



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)
```

Weight initialization

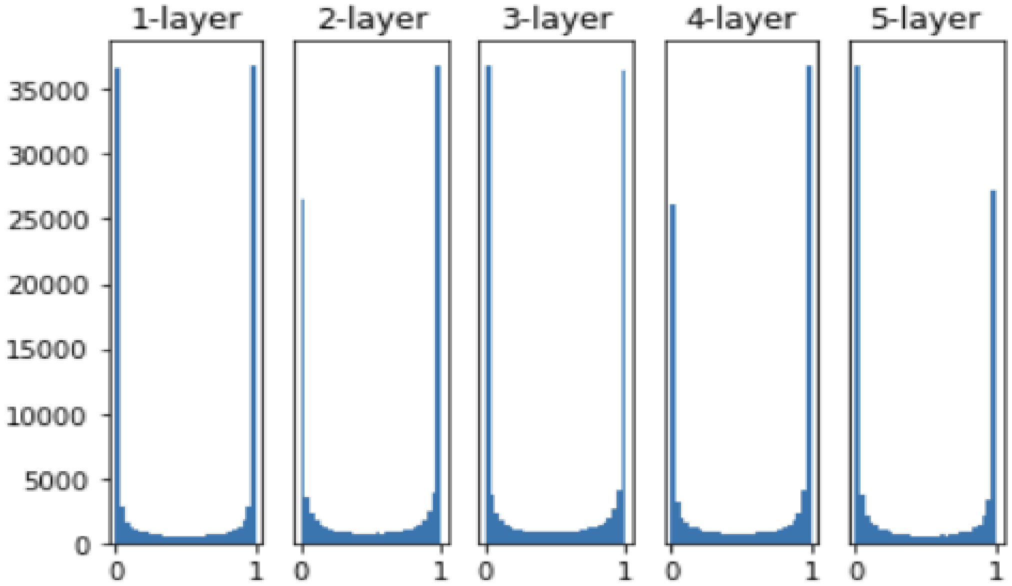
0에 가까운
random number로 초기화

-> 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

Weight initialization

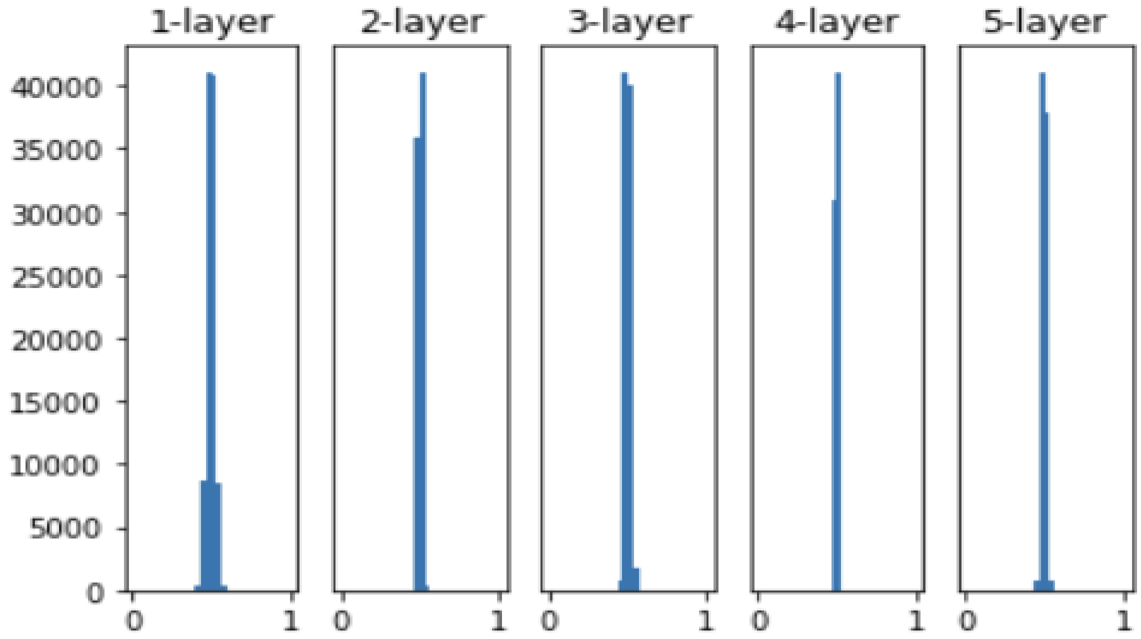


Vanishing gradient 발생

Weight initialization

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

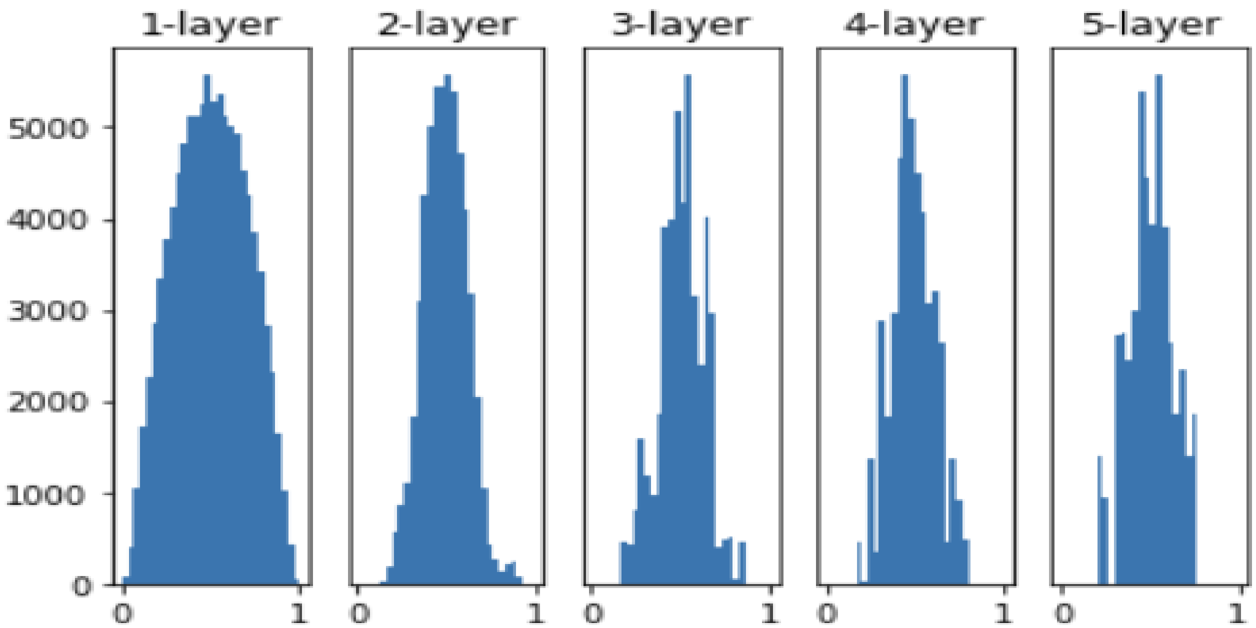
표준편차가 0.01인 정규분포



Xavier initialization

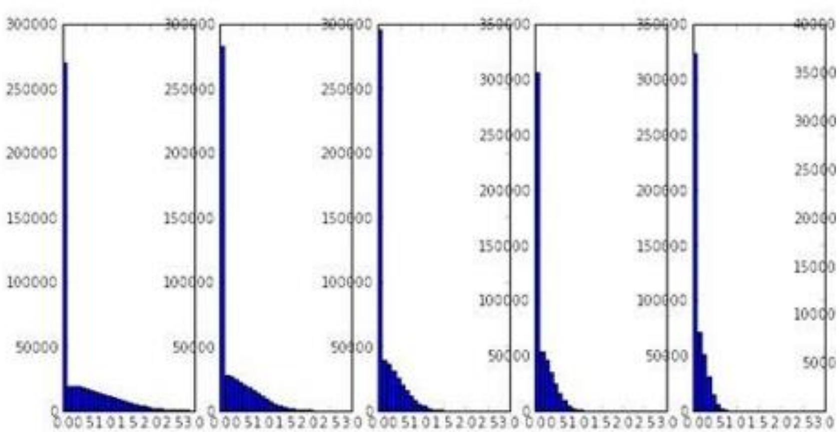
```python
node_num = 100 # 앞 층의 노드 수
w = np.random.randn(node_num, node_num) / np.sqrt(node_num)
```

node_in, node_out node_in

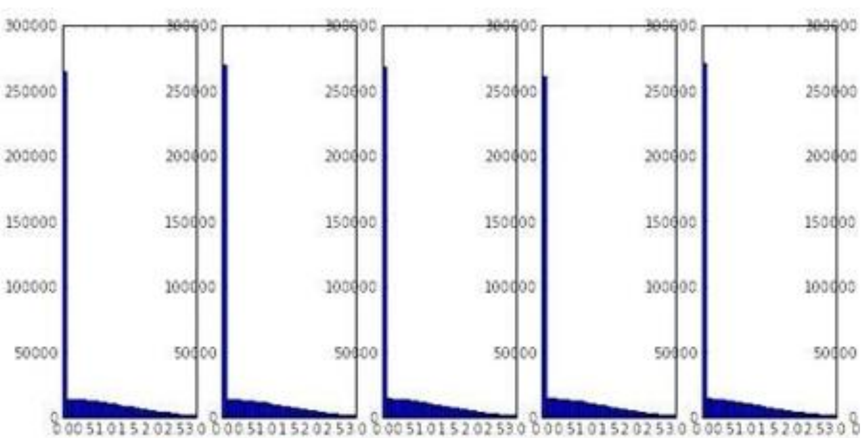


He initialization

```python
#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

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

Batch Normalization

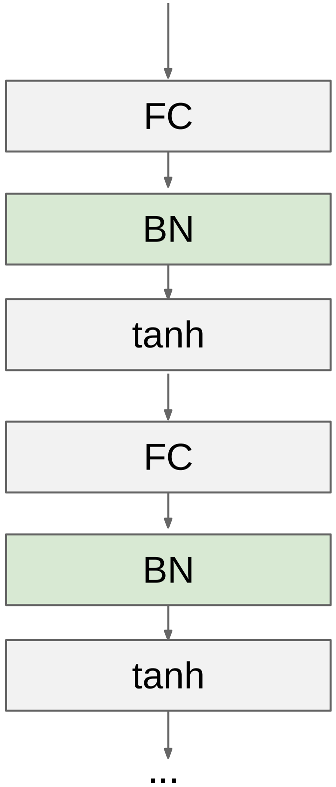
Mini Batch 단위로 Normalization 수행

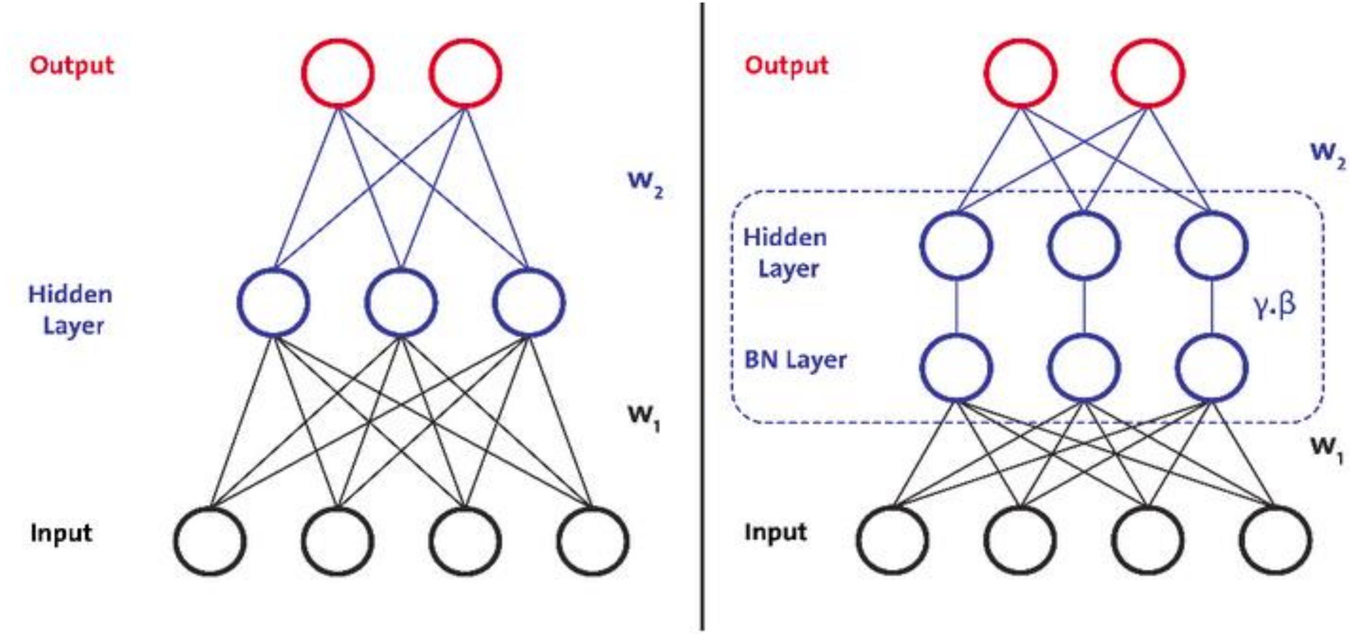
차원의 입력 각각(x^(1)~x^(d))에 대하여 아래 연산(정규화)을 수행

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

☞ 정규화 결과, 평균은 0, 분산은 1이 됨

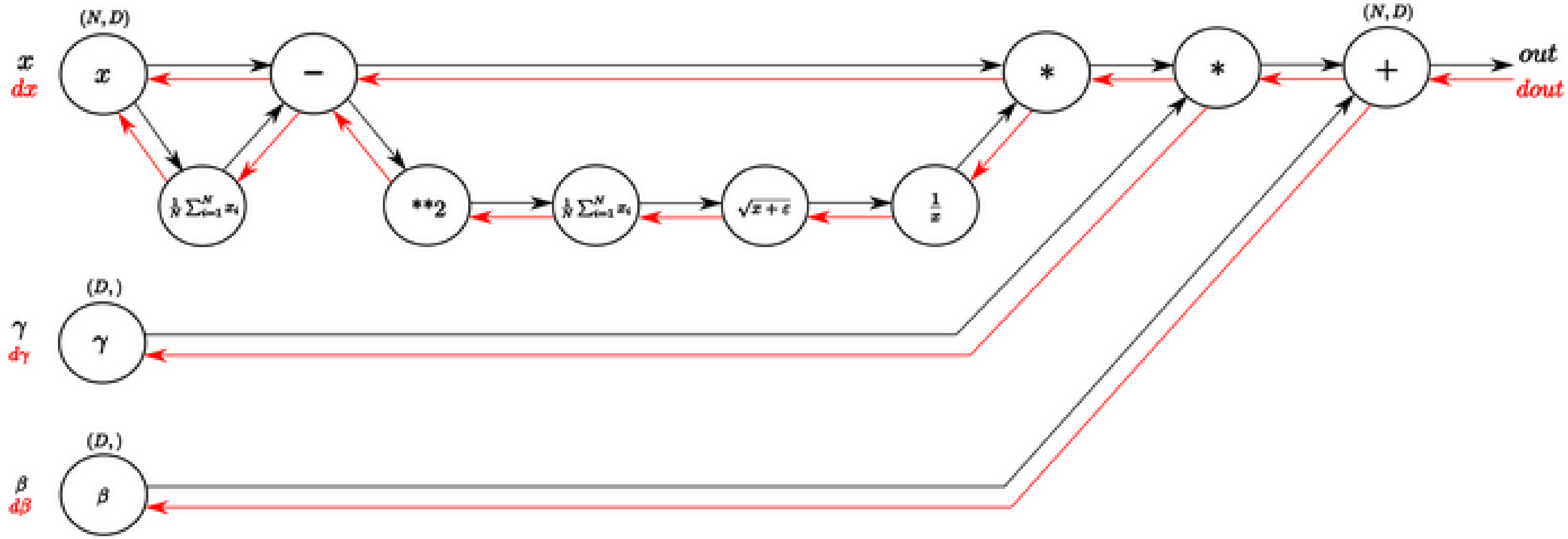
Batch Normalization





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

Batch Normalization



Batch Normalization

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

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

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

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

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

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

Batch Normalization

정규화

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

Scale/Shift 변환 수행

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

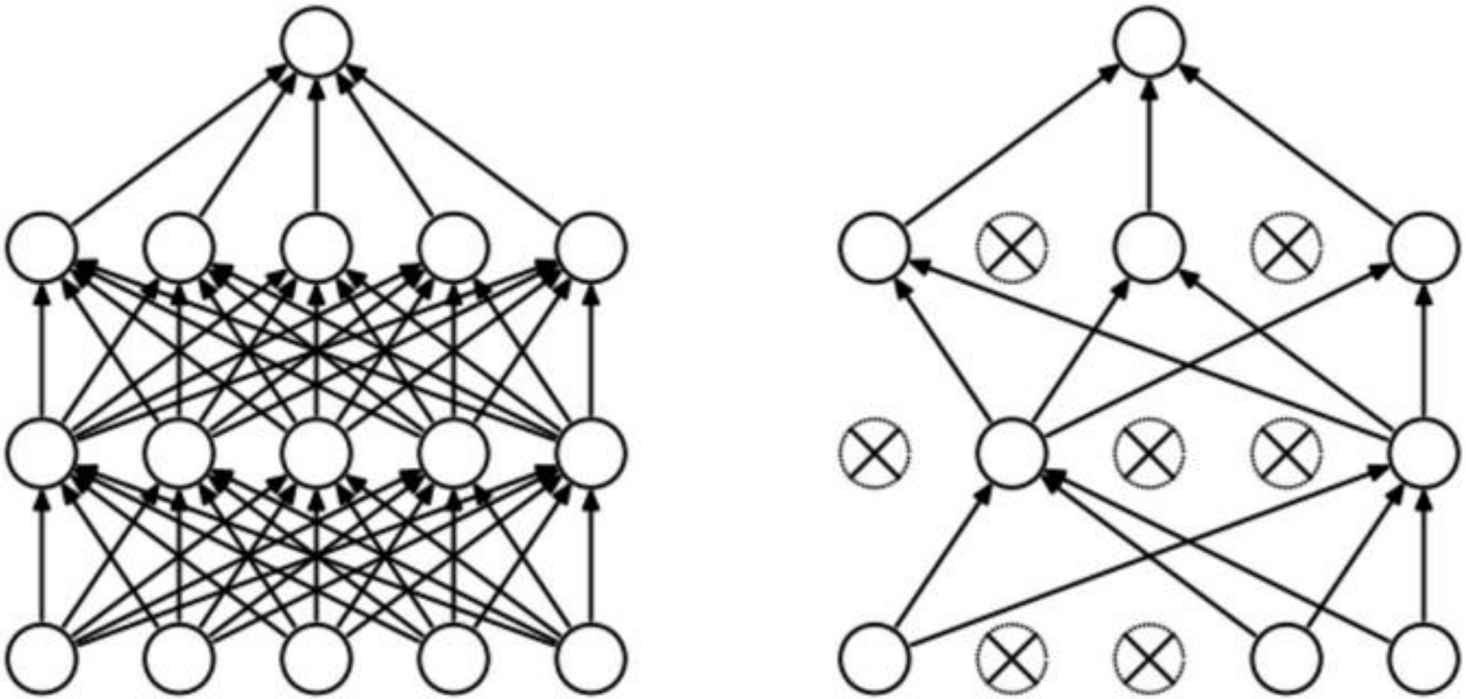
Note, the network can learn:

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

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

to recover the identity mapping.

Dropout: Prevent Overfitting



(a) Standard Neural Net

(b) After applying dropout.

Dropout: Prevent Overfitting

Hidden Layer의 Neuron 임의 누락

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

→ Overfitting 방지

Dropout: Prevent Overfitting

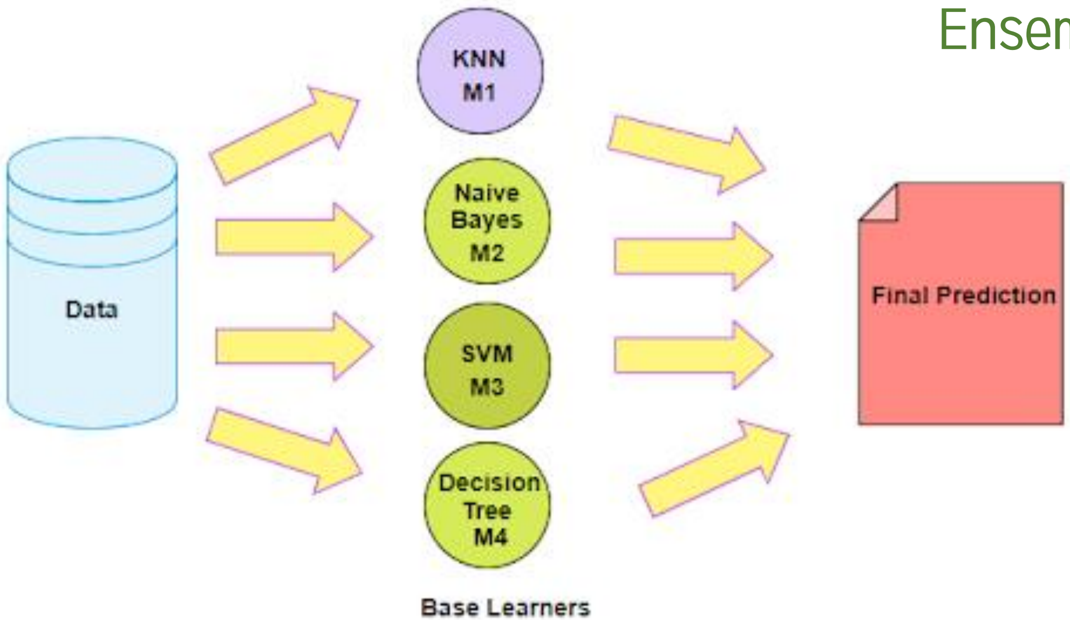
Forward pass 과정에서 일부 뉴런의 값을 임의로 0으로 만들어 구현

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

```
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
```

Dropout: Prevent Overfitting

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



Babysitting the Learning Process

1. Preprocess the data

2. Choose the architecture

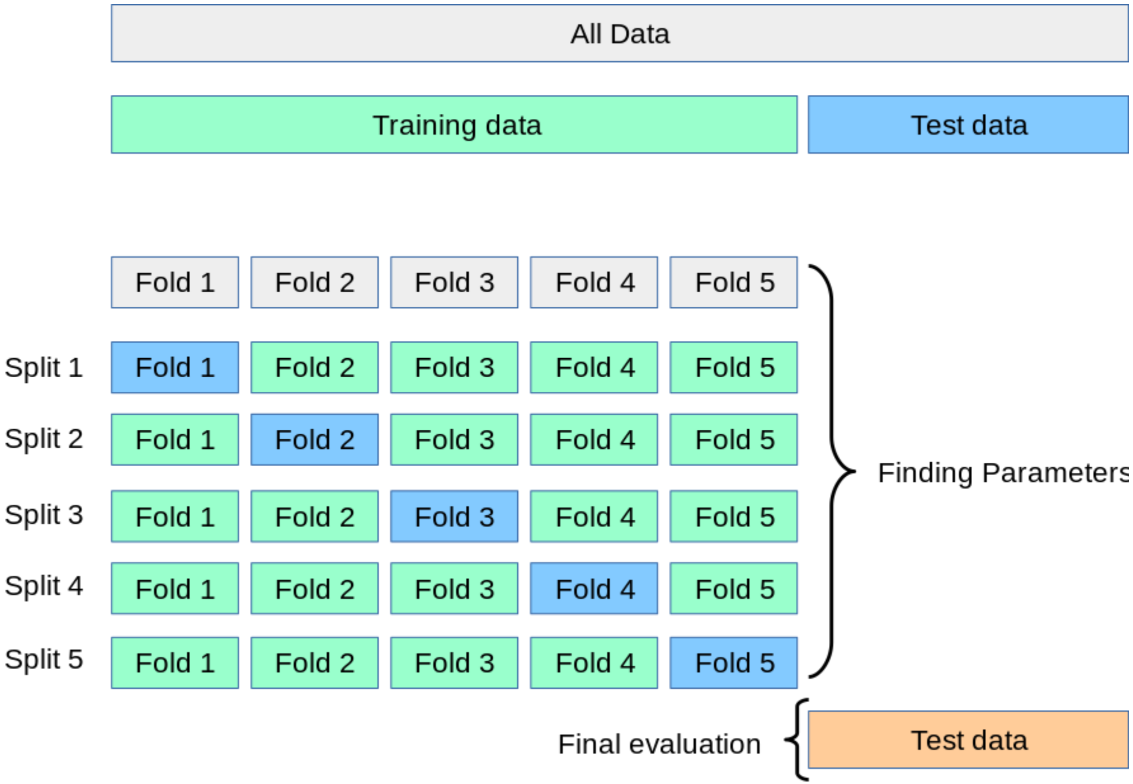
- Check the loss is reasonable

3. Train

- Check overfitting
- Start with small regularization
- Find learning rate

If the learning rate is too low, loss not going down

Hyperparameter Optimization



Cross-Validation Strategy

Train at your Train Set, Validate at your Validation Set

Hyperparameter Optimization

Few epochs

→ Longer Running Time & Finer Search

Hyperparameter Optimization

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

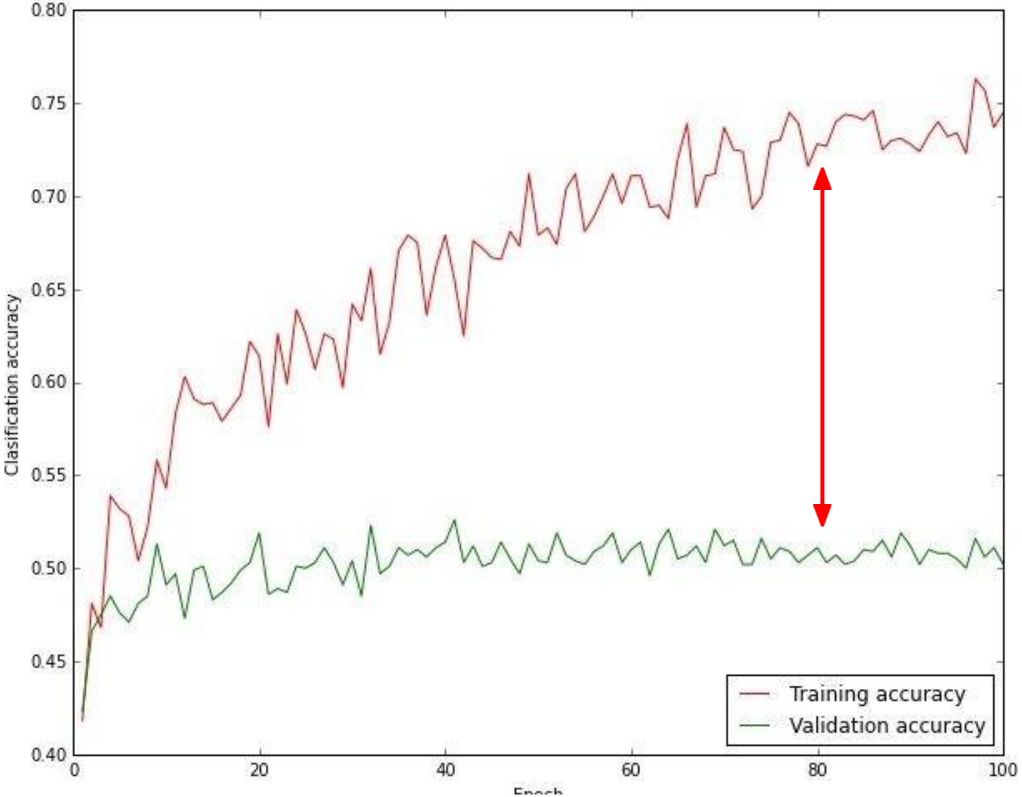
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

Hyperparameter Optimization

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons

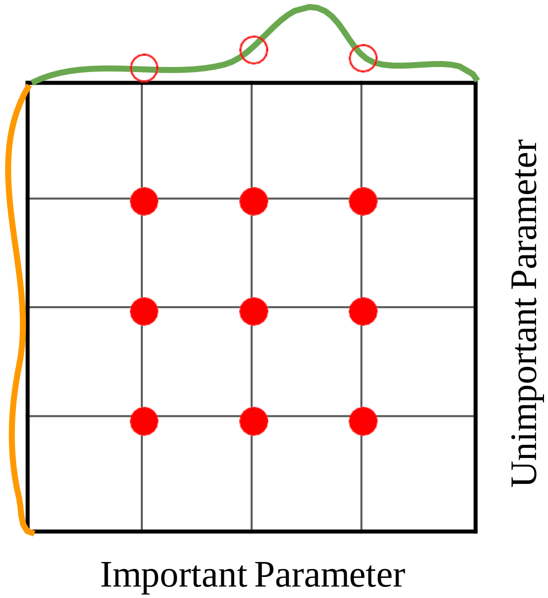
Hyperparameter Optimization

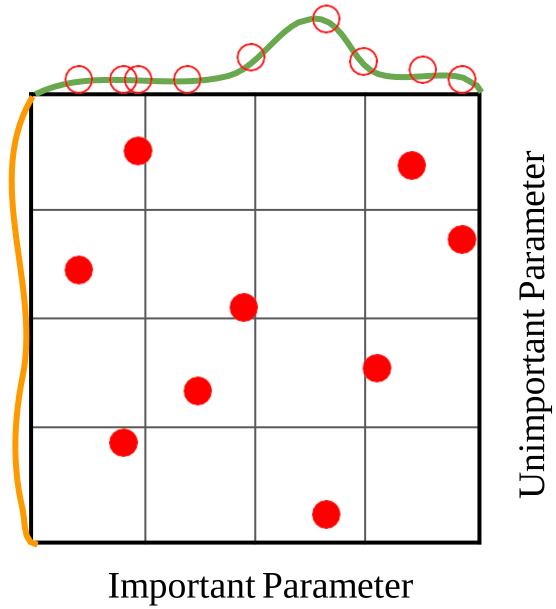


Big Gap
Val – Train
: Overfitting?

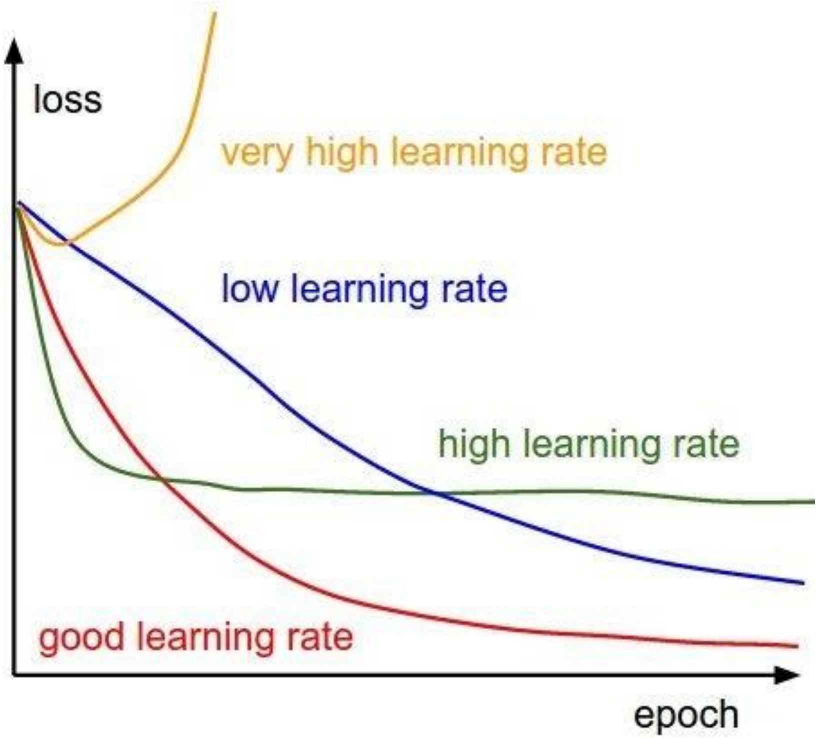
Hyperparameter Optimization

Random Search: samples ↑, signals ↑





Hyperparameter Optimization



Hyperparameter

1. Network Architecture
2. Learning rate, decay schedule, update type
3. Regularization

감사합니다

Q&A