Weight initialization

Overview

우리가 배워야 할 것

저번주까지는 **Activate Function**을 배웠다
오늘은 **Weight Initialization**에 대해 배울 것

```
learning_rate = 0.000001

w1 = np.random.uniform(low=-0.058, high=+0.058, size=(784, 1000))
w2 = np.random.uniform(low=-0.077, high=+0.077, size=(1000, 10))

num_epoch = 100

for epoch in range(num_epoch):
    # Forward propagation
    z1 = X_train.dot(w1)
    a1 = sigmoid(z1)
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)

    # Backpropagation
    d2 = a2 - y_train_hot
    d1 = d2.dot(w2.T) * a1 * (1 - a1)

    w2 = w2 - learning_rate * a1.T.dot(d2)
    w1 = w1 - learning_rate * X_train.T.dot(d1)
```

앞으로 우리가 배워야 할 것

저번주까지는 Activate Function을 배웠다
오늘은 Weight Initialization에 대해 배울 것

```
learning_rate = 0.000001

w1 = np.random.uniform(low=-0.058, high=+0.058, size=(784, 1000))
w2 = np.random.uniform(low=-0.077, high=+0.077, size=(1000, 10))

num_epoch = 100

for epoch in range(num_epoch):
    # Forward propagation
    z1 = X_train.dot(w1)
    a1 = sigmoid(z1)
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)

    # Backpropagation
    d2 = a2 - y_train_hot
    d1 = d2.dot(w2.T) * a1 * (1 - a1)

    w2 = w2 - learning_rate * a1.T.dot(d2)
    w1 = w1 - learning_rate * X_train.T.dot(d1)
```

Weight Initialization

Activation Function

Automatic Differentiation

Optimizer

앞으로 우리가 배워야 할 것

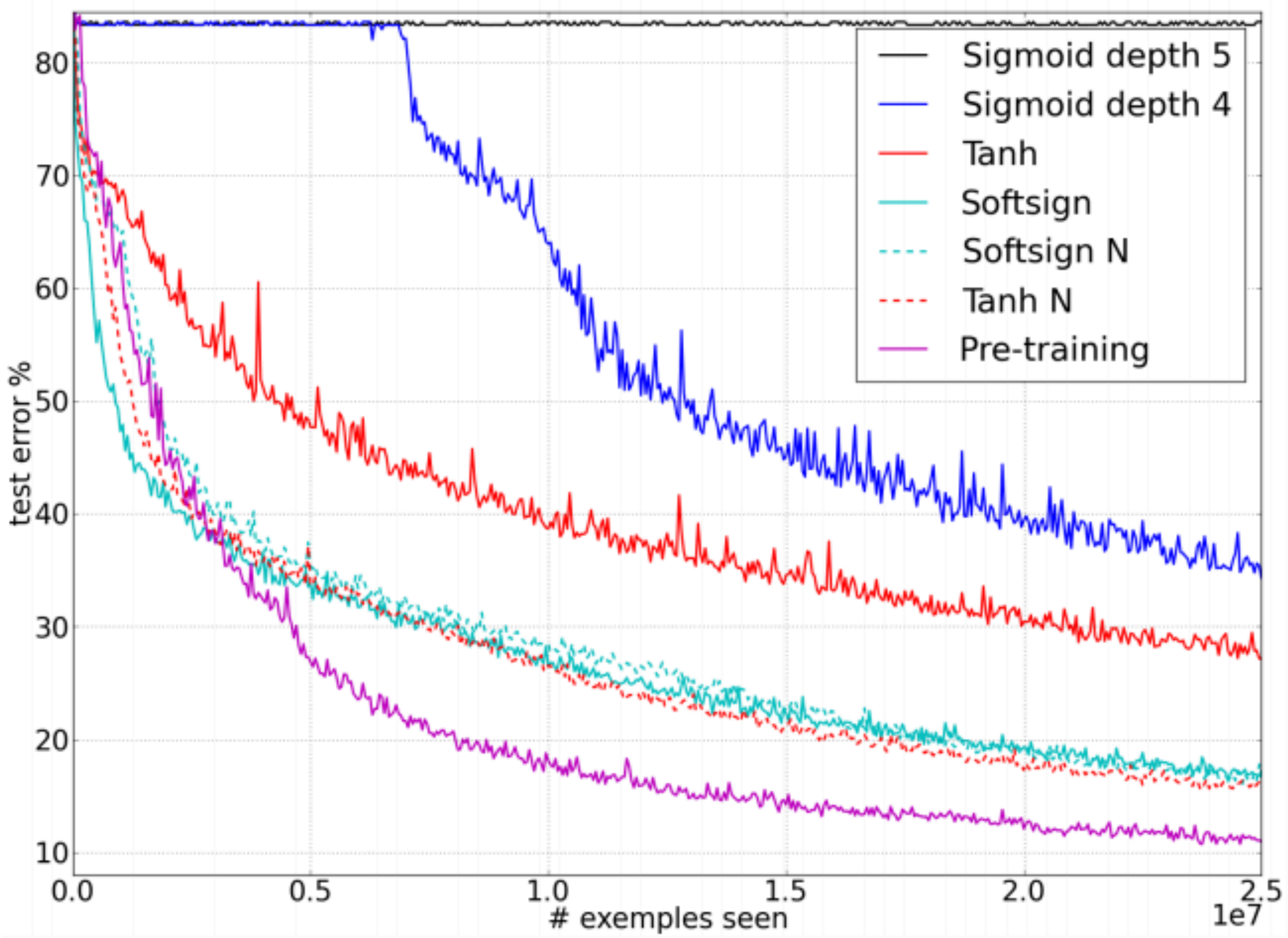
저번주까지는 Activate Function을 배웠다
오늘은 Weight Initialization에 대해 배울 것

Weight Initialization

```
learning_rate = 0.000001

w1 = np.random.uniform(low=-0.058, high=+0.058, size=(784, 1000))
w2 = np.random.uniform(low=-0.077, high=+0.077, size=(1000, 10))

num_epoch = 100

for epoch in range(num_epoch):
    # Forward propagation
    z1 = X_train.dot(w1)
    a1 = sigmoid(z1)
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)

    # Backpropagation
    d2 = a2 - y_train_hot
    d1 = d2.dot(w2.T) * a1 * (1 - a1)

    w2 = w2 - learning_rate * a1.T.dot(d2)
    w1 = w1 - learning_rate * X_train.T.dot(d1)
```

Weight Initialization?

weight를 어떻게 초기화하냐에 따라서 결과가 전혀 달라진다
굉장히 간단한 원리를 통해 모델의 성능을 확연하게 끌어올릴 수 있다

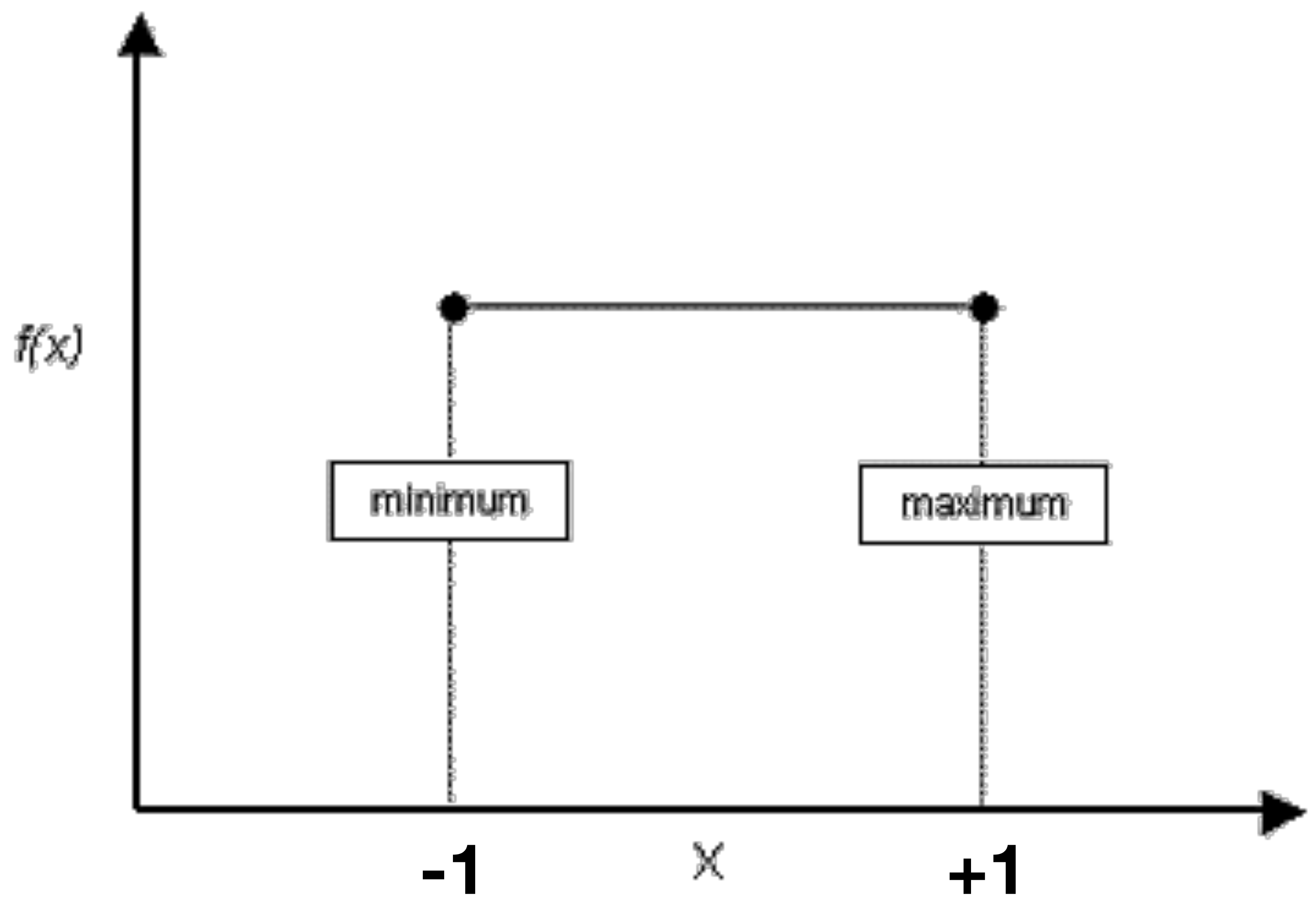


같은 모델이라도 weight를 어떻게 주냐에 따라
결과가 확연히 달라진다

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

Small Random Number

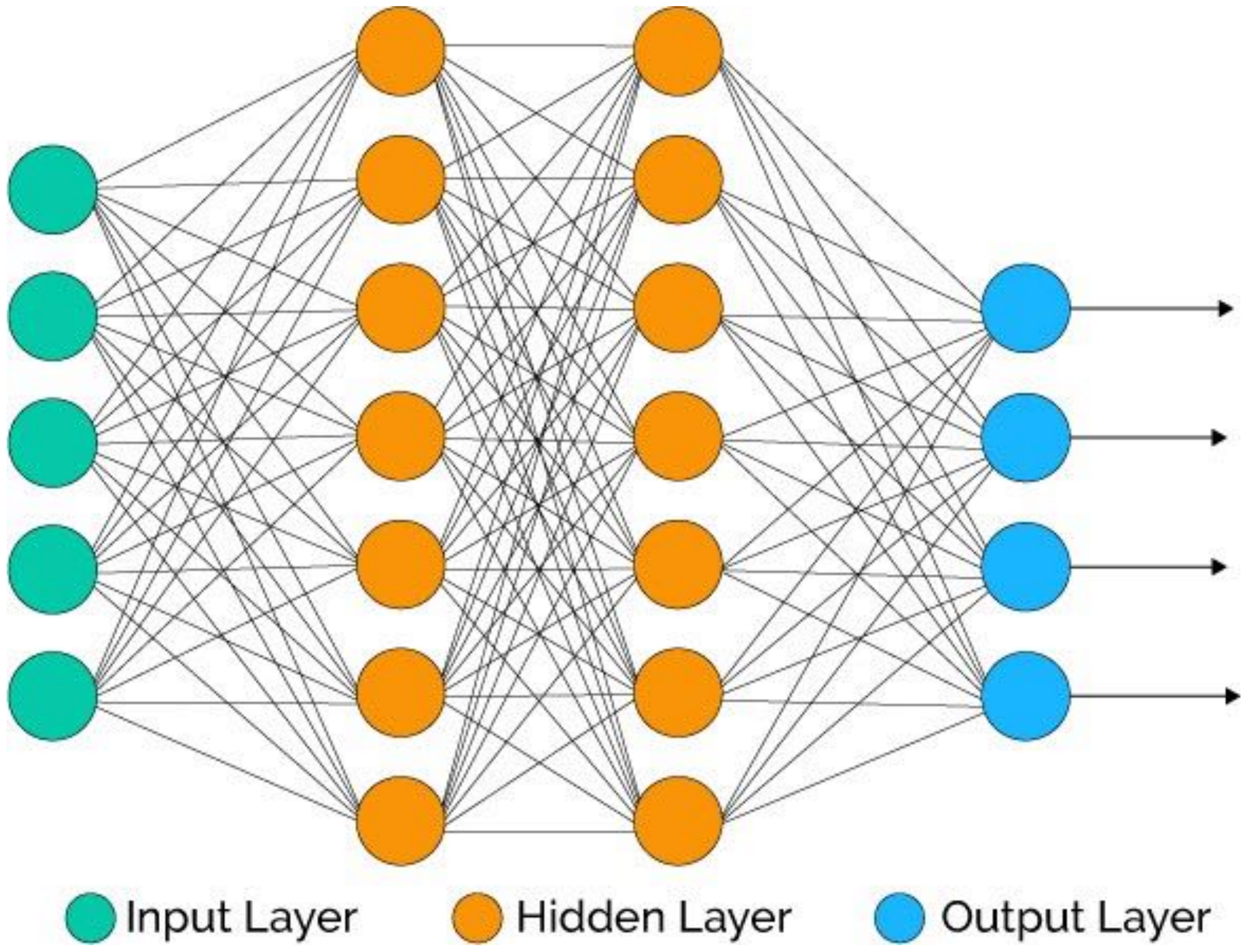
적당한 random값을 준다. 가장 기본적인 방식이자 우리가 지금까지 해왔던 방식
min/max (내지는 std)를 잘 주지 않으면 제대로 동작하지 않는다




```
w1 = np.random.uniform(low=-1.0, high=1.0, size=(64, 100))
w2 = np.random.uniform(low=-1.0, high=1.0, size=(100, 100))
w3 = np.random.uniform(low=-1.0, high=1.0, size=(100, 10))
```

왜 Small Random Number는 제대로 동작하지 않는가?

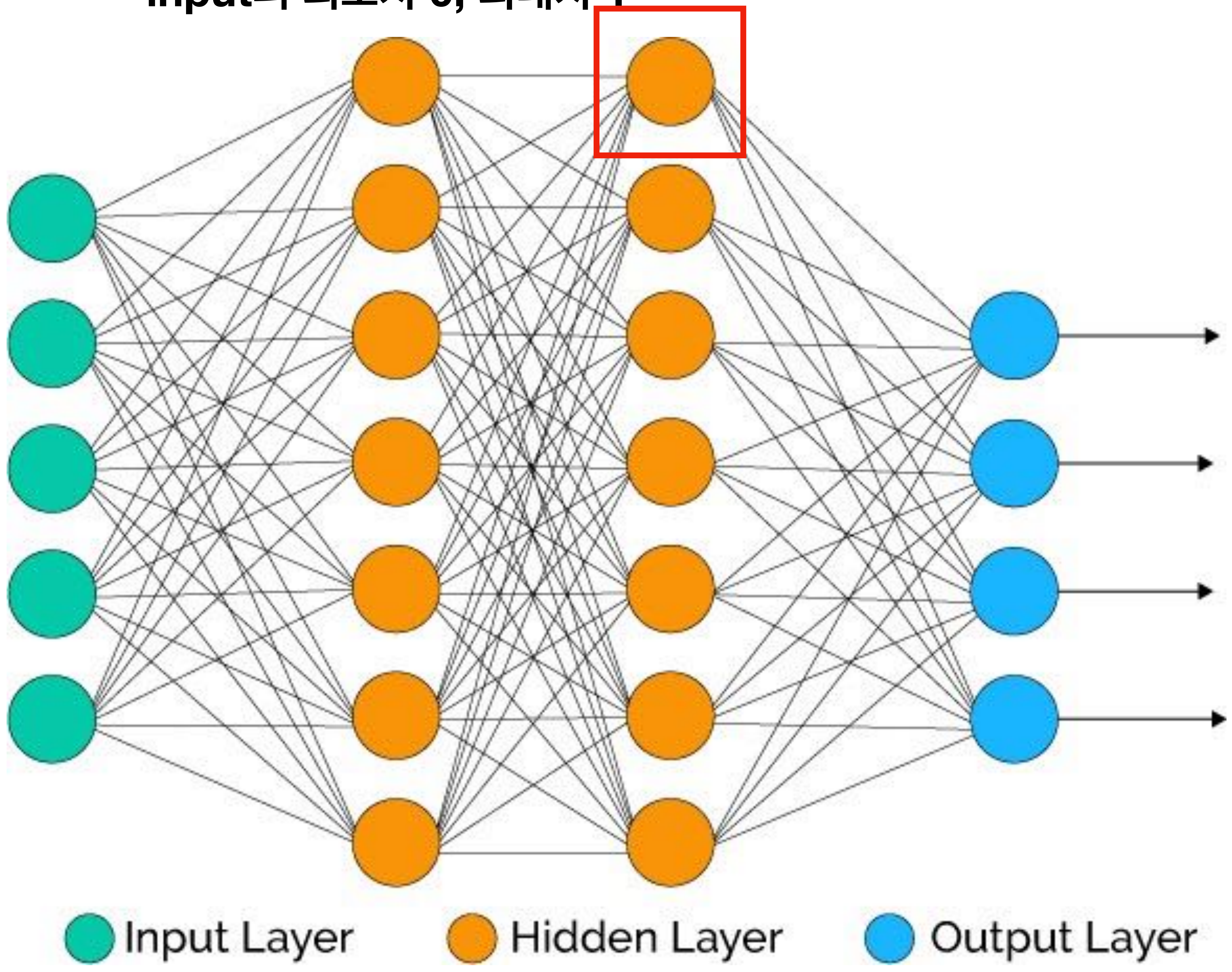
모든 레이어는 이전 레이어의 결과값을 더한 뒤 activate 함수에 넣는데,
이전 레이어의 결과가 크면 saturate 되어버릴 가능성이 높음



왜 Small Random Number는 제대로 동작하지 않는가?

모든 레이어는 이전 레이어의 결과값을 더한 뒤 activate 함수에 넣는데,
이전 레이어의 결과가 크면 saturate 되어버릴 가능성이 높음

뉴런의 갯수 = 100개
input의 최소치 0, 최대치 1



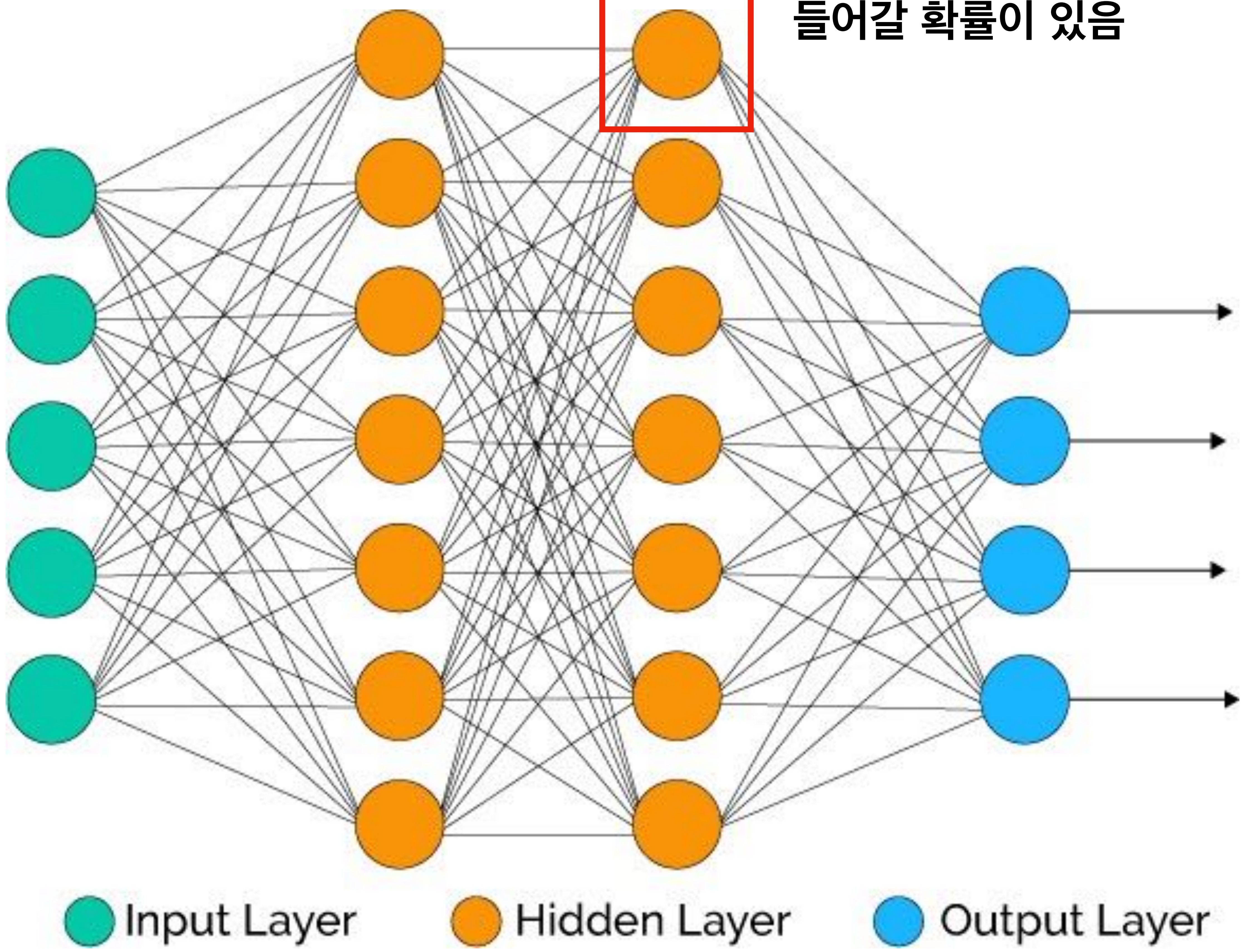
weight
input의 최소치 -1, 최대치 1

왜 Small Random Number는 제대로 동작하지 않는가?

모든 레이어는 이전 레이어의 결과값을 더한 뒤 activate 함수에 넣는데,
이전 레이어의 결과가 크면 saturate 되어버릴 가능성이 높음

뉴런의 갯수 = 100개
input의 최소치 0, 최대치 1

여기에 최대 -100 ~ 100이
들어갈 확률이 있음



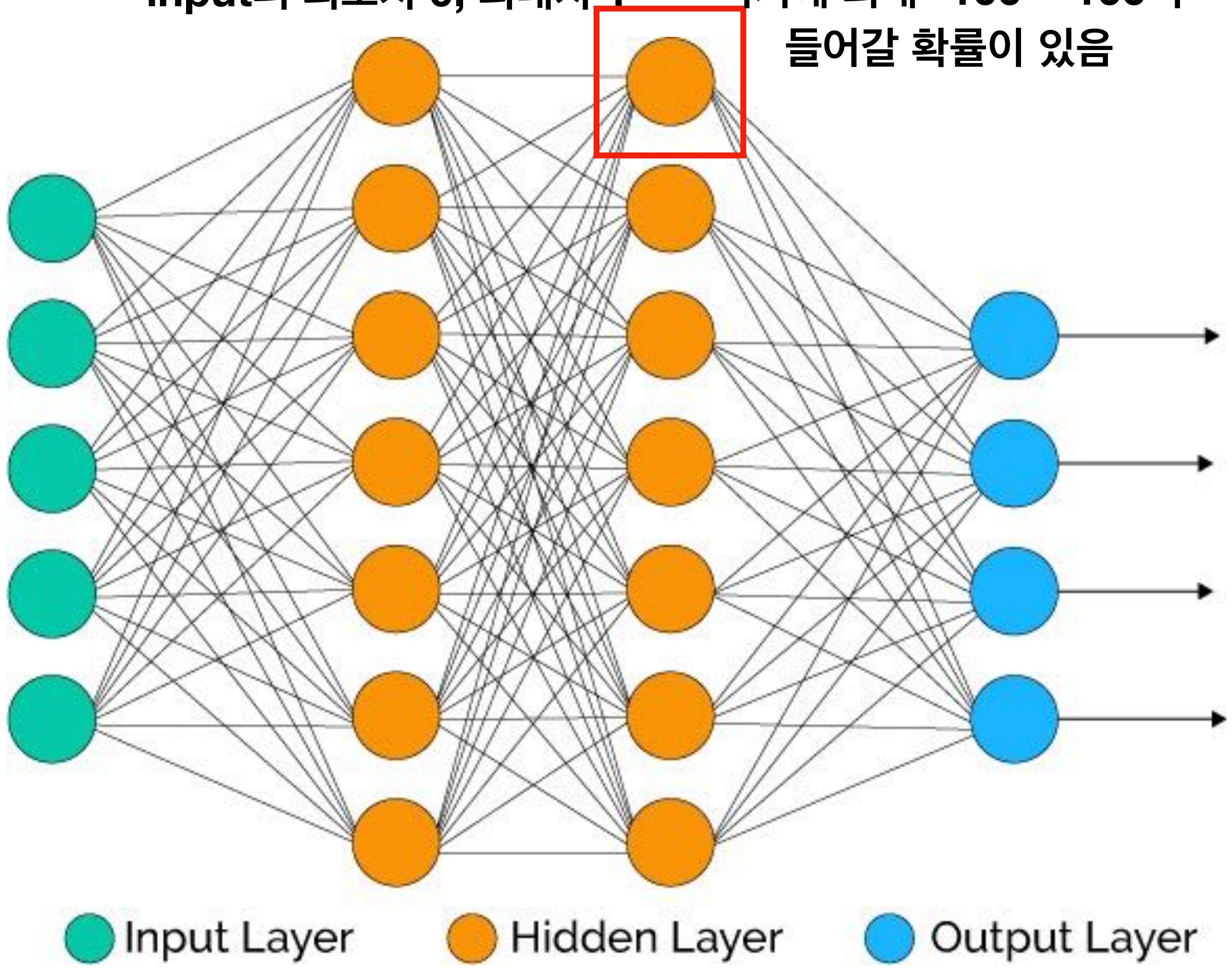
weight
input의 최소치 -1, 최대치 1

왜 Small Random Number는 제대로 동작하지 않는가?

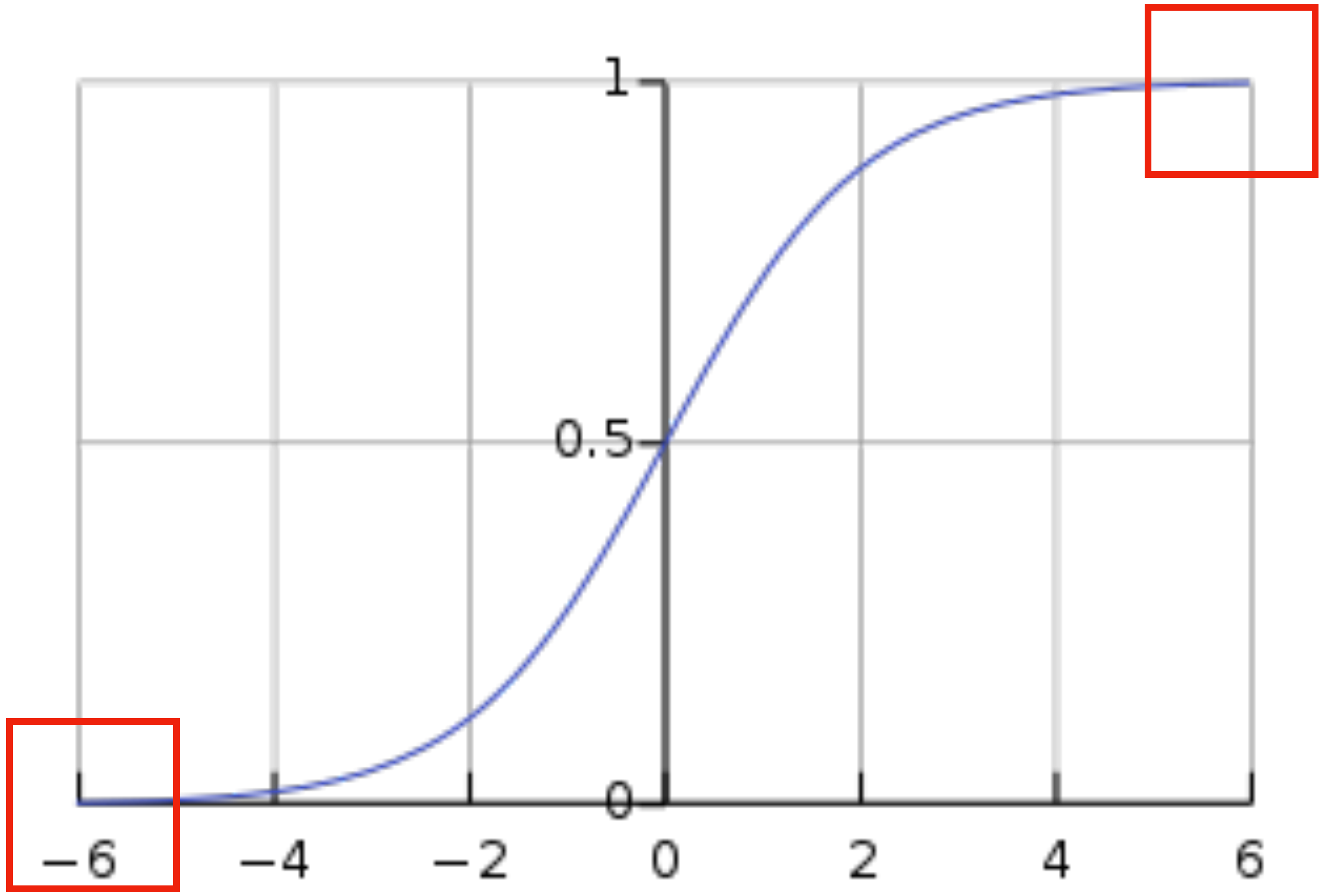
모든 레이어는 이전 레이어의 결과값을 더한 뒤 activate 함수에 넣는데,
이전 레이어의 결과가 크면 saturate 되어버릴 가능성이 높음

뉴런의 갯수 = 100개
input의 최소치 0, 최대치 1

여기에 최대 -100 ~ 100이
들어갈 확률이 있음



weight
input의 최소치 -1, 최대치 1



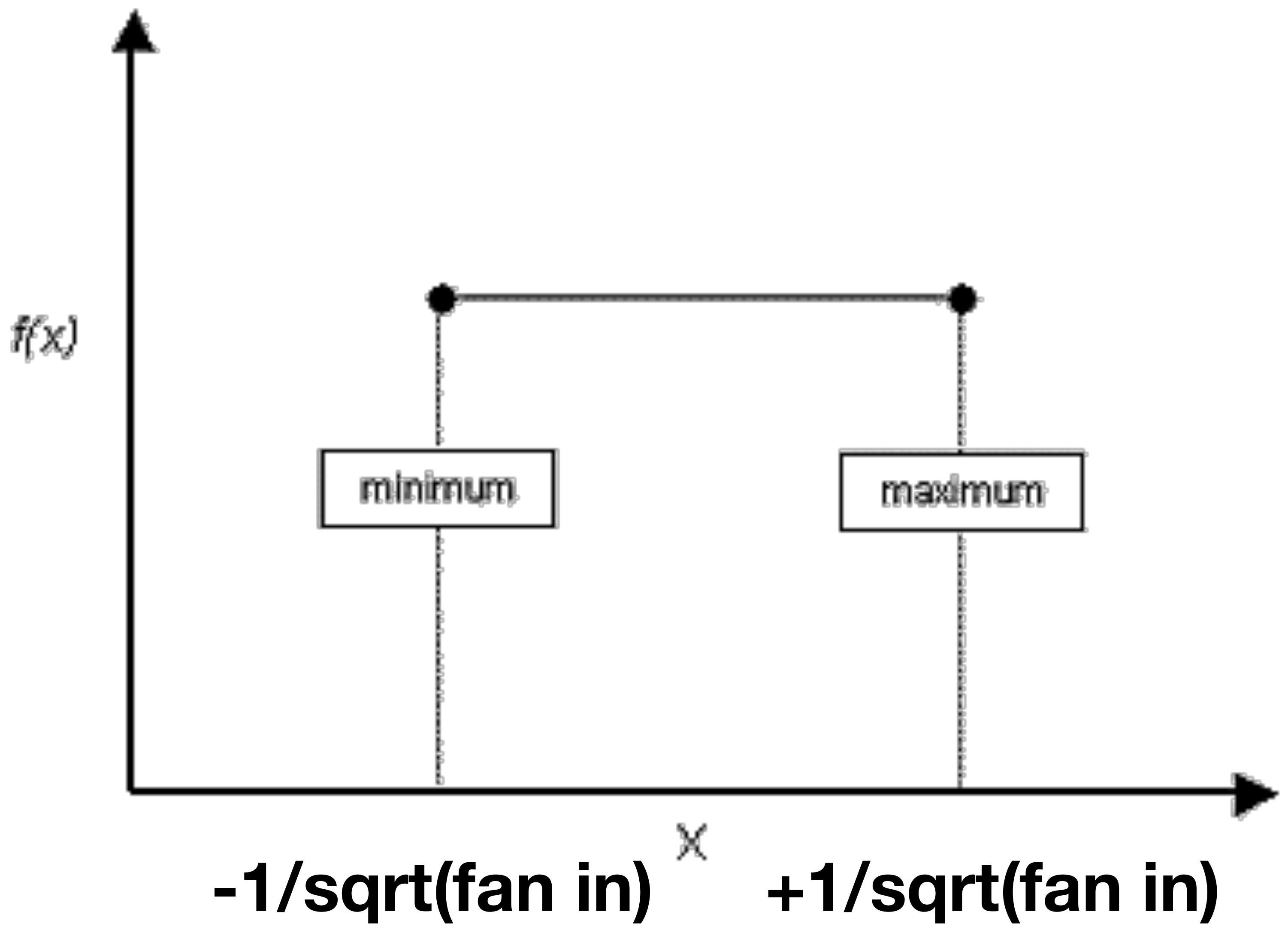
-100이나 100을 activate function에 집어넣으면
반드시 saturate하게 되어있다

Solution 1 - input size(fan in)을 기준으로 normalize하자!

input size(이하 fan in)가 saturation에 큰 영향이 있다면, input을 기준으로 normalize하면 되지 않을까?
라는 가정 하에 1/sqrt(fan in)으로 초기화한다

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right],$$

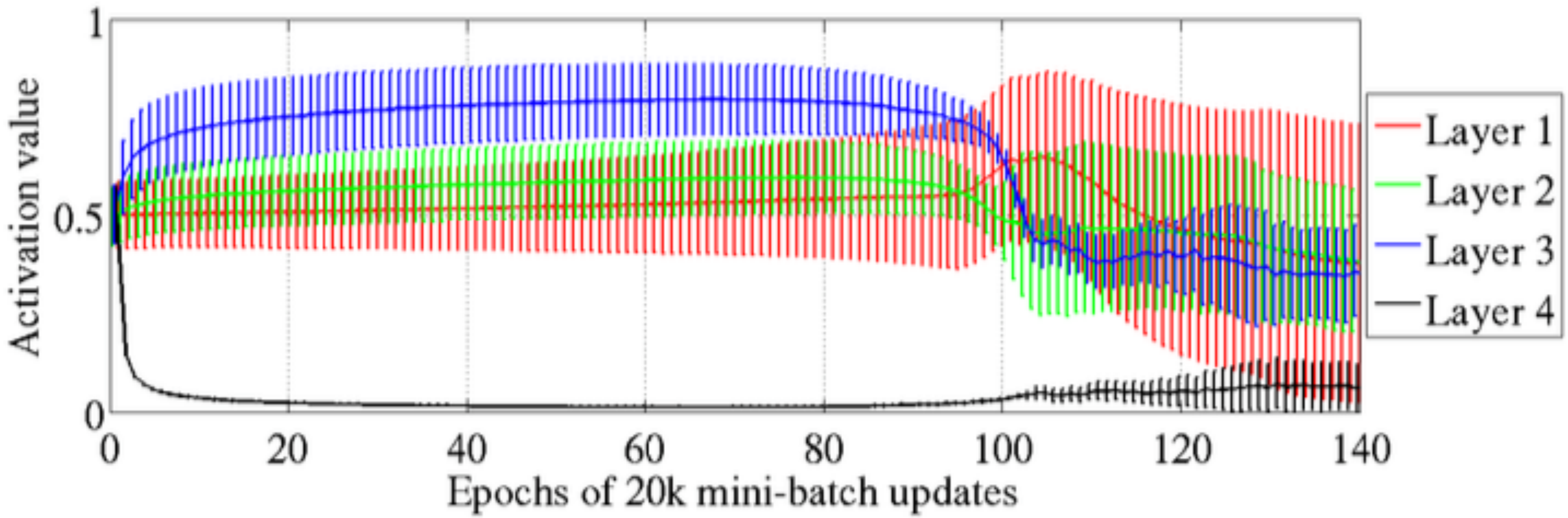
```
a = 1.0 / np.sqrt(64)

w = np.random.uniform(low=-a, high=+a, size=(64, 100))
```



Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

Solution 1 - input size(fan in)을 기준으로 normalize하자!

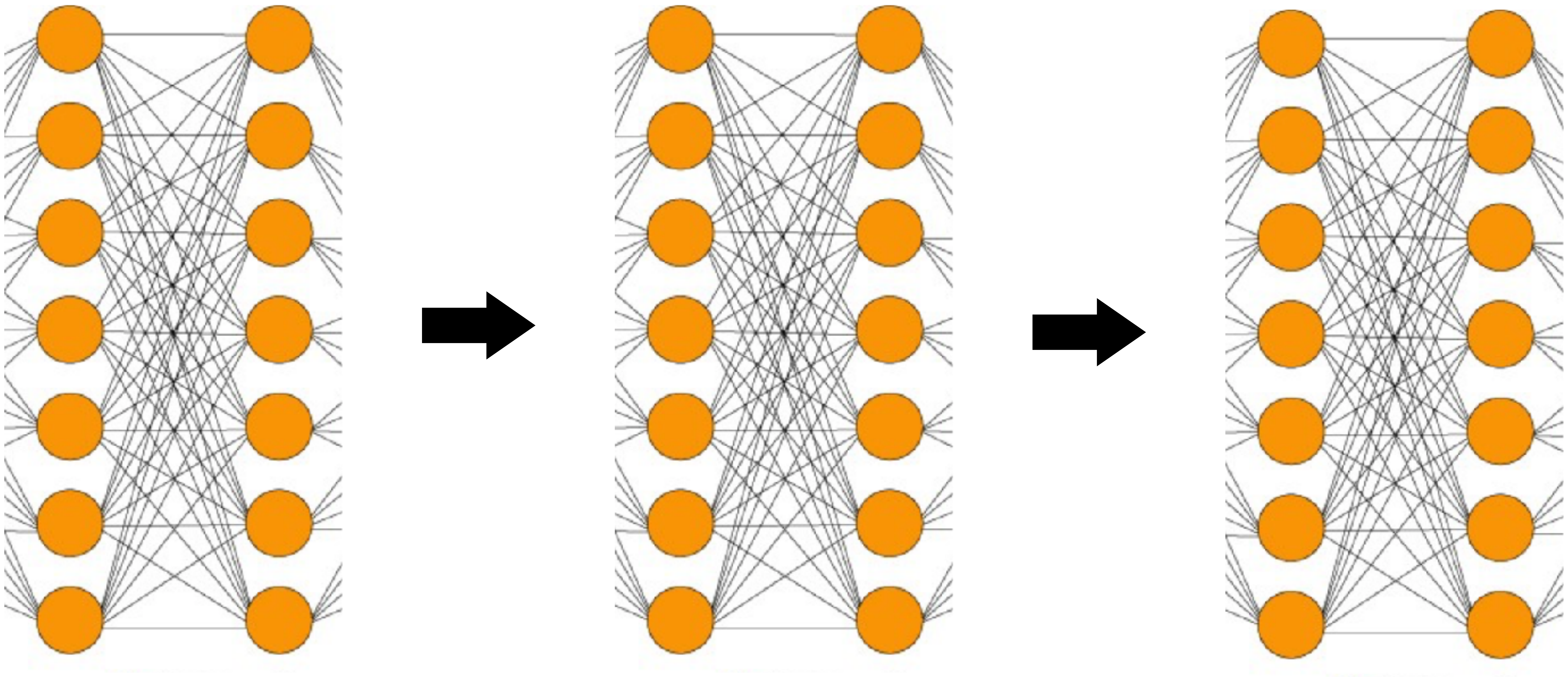
레이어마다 weight의 update가 크게 바뀌는 것을 알 수 있다
이렇게 되면 레이어 4 이상으로 더 쌓는 것이 불가능해진다.



Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

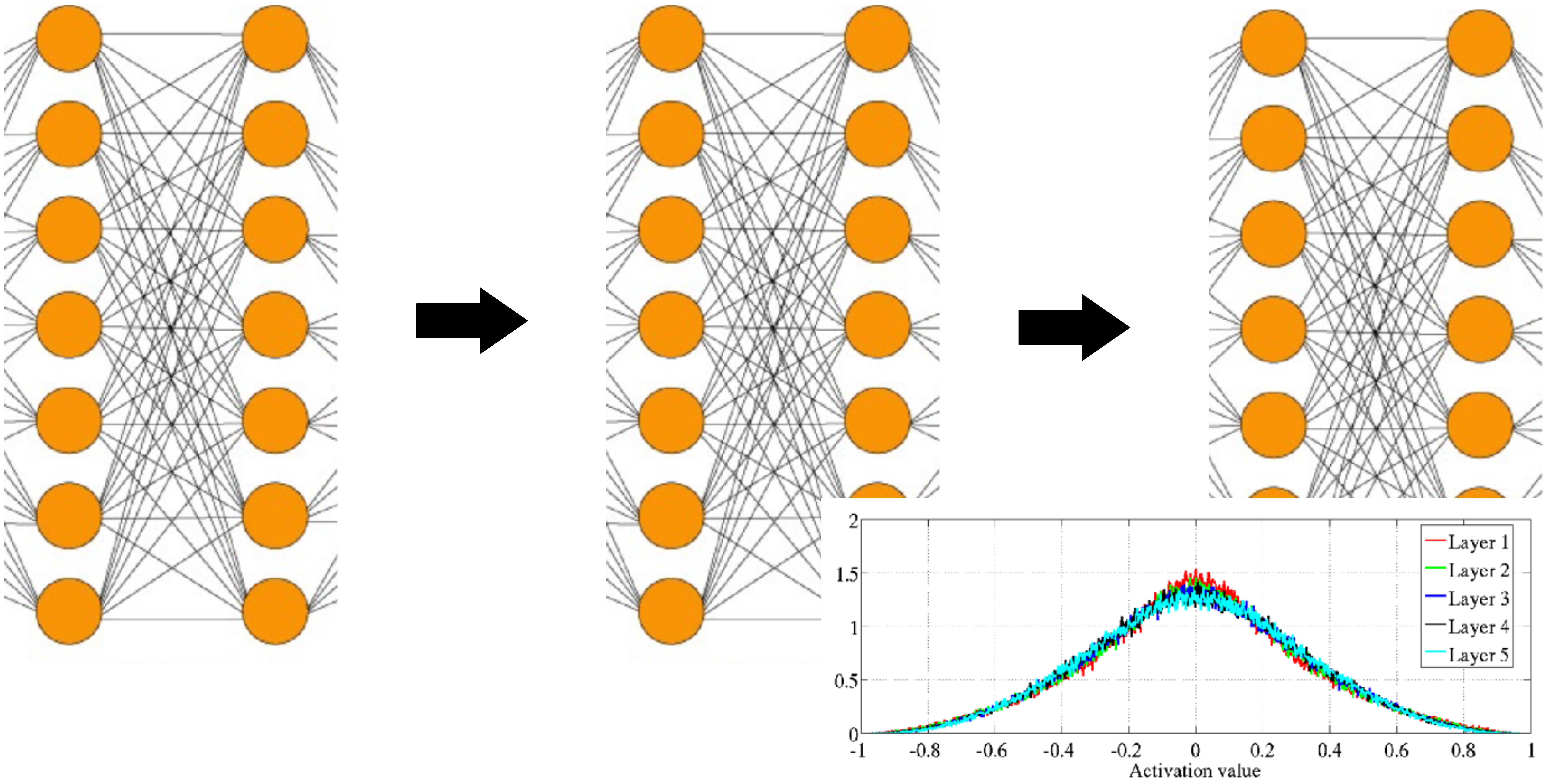
레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.



Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

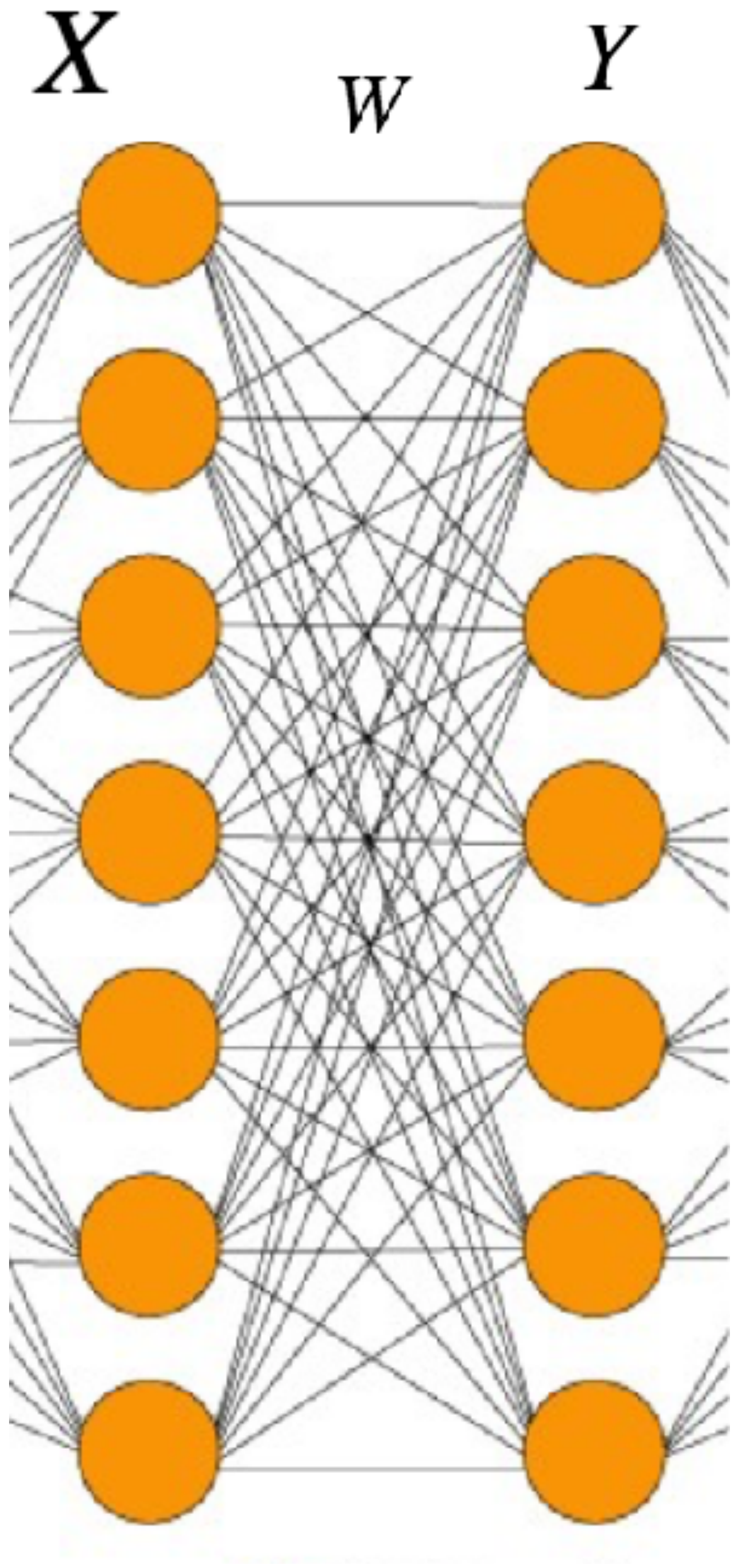
레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

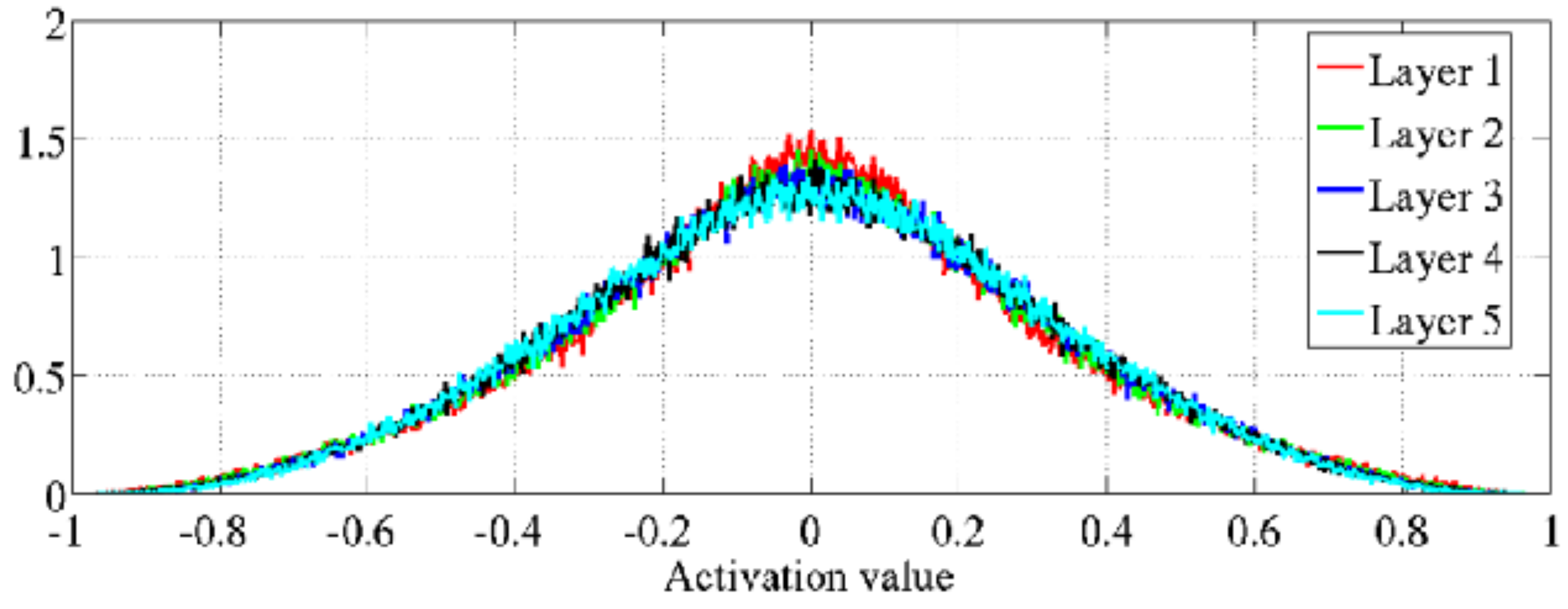


Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.





Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

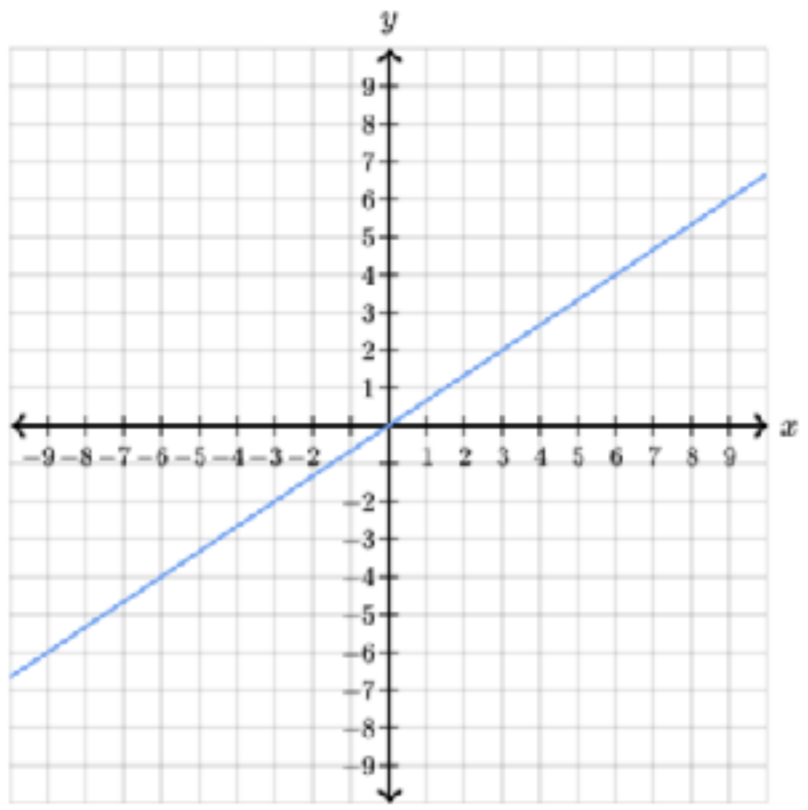
$$Z = W \cdot X$$

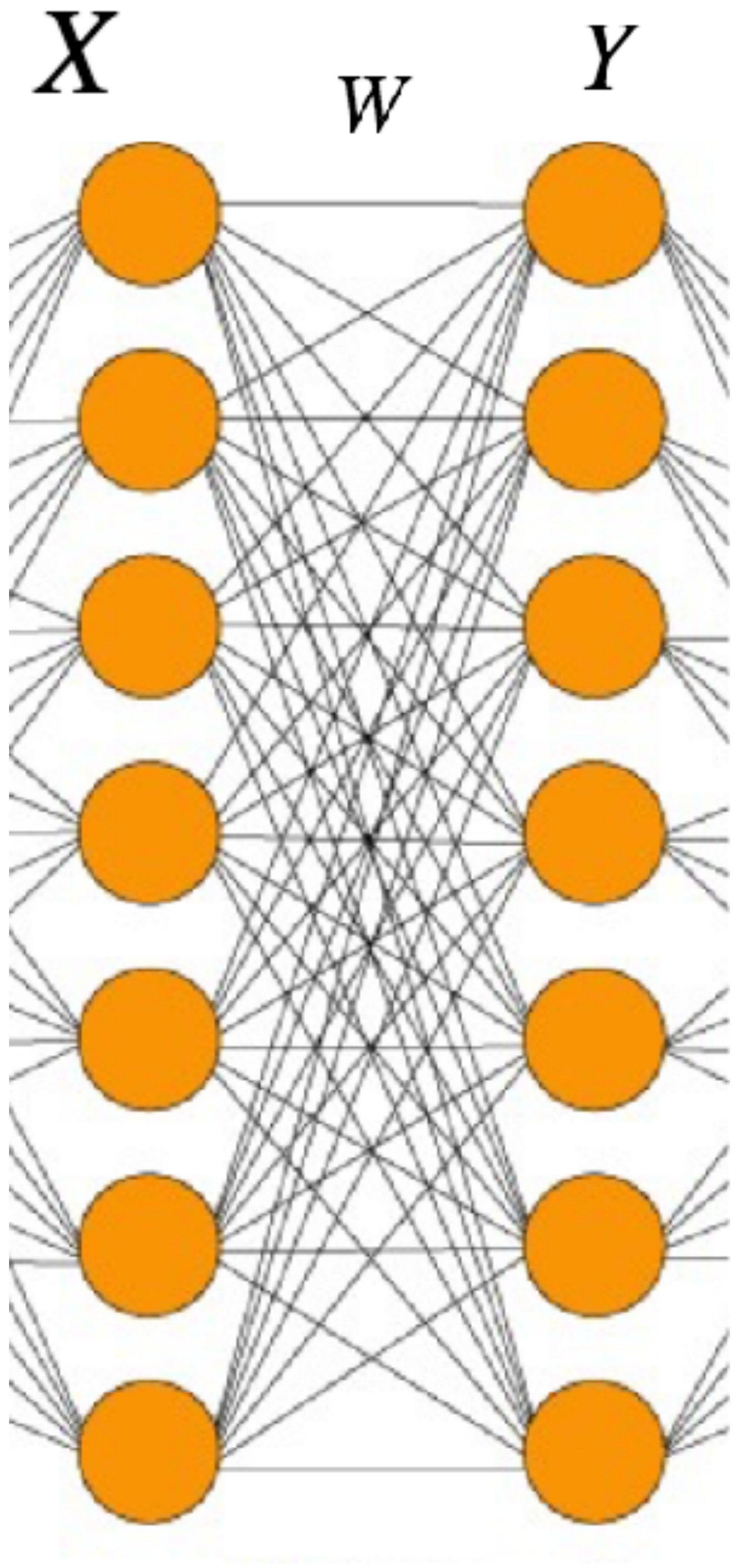
$$Y = \boxed{f(Z)}$$

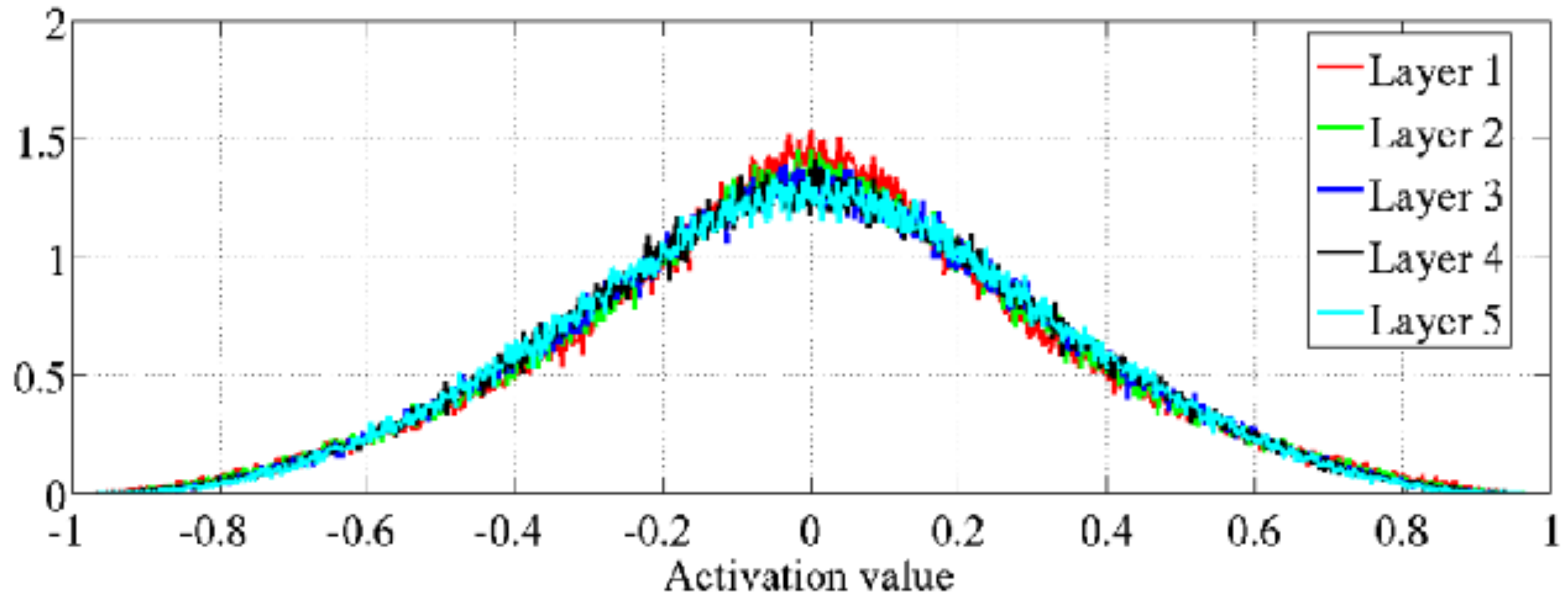
activation function

$$f(x) = x$$

$$f'(x) = 1$$







Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

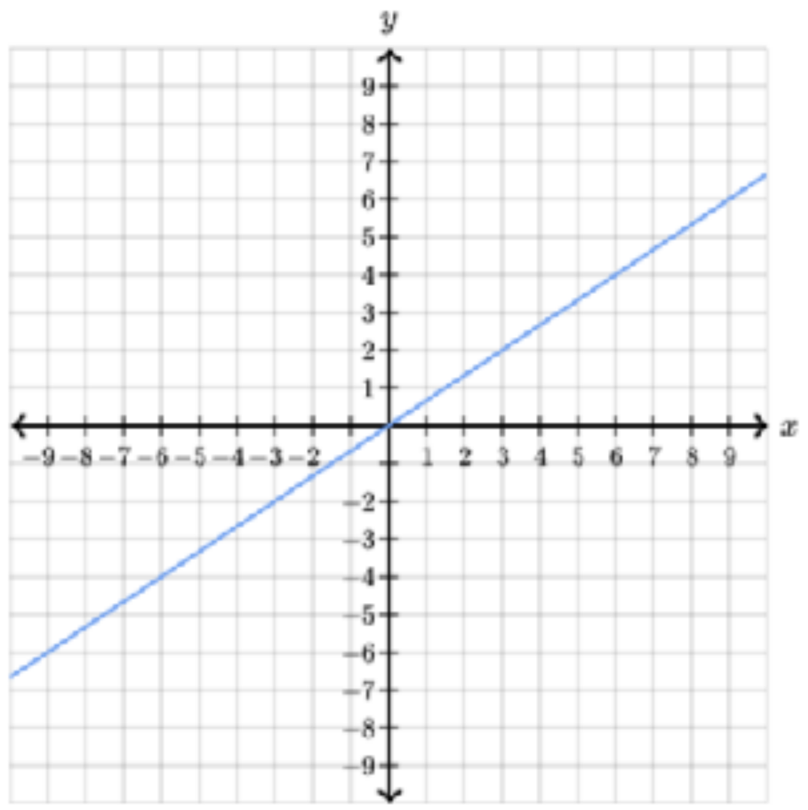
$$Z = W \cdot X$$

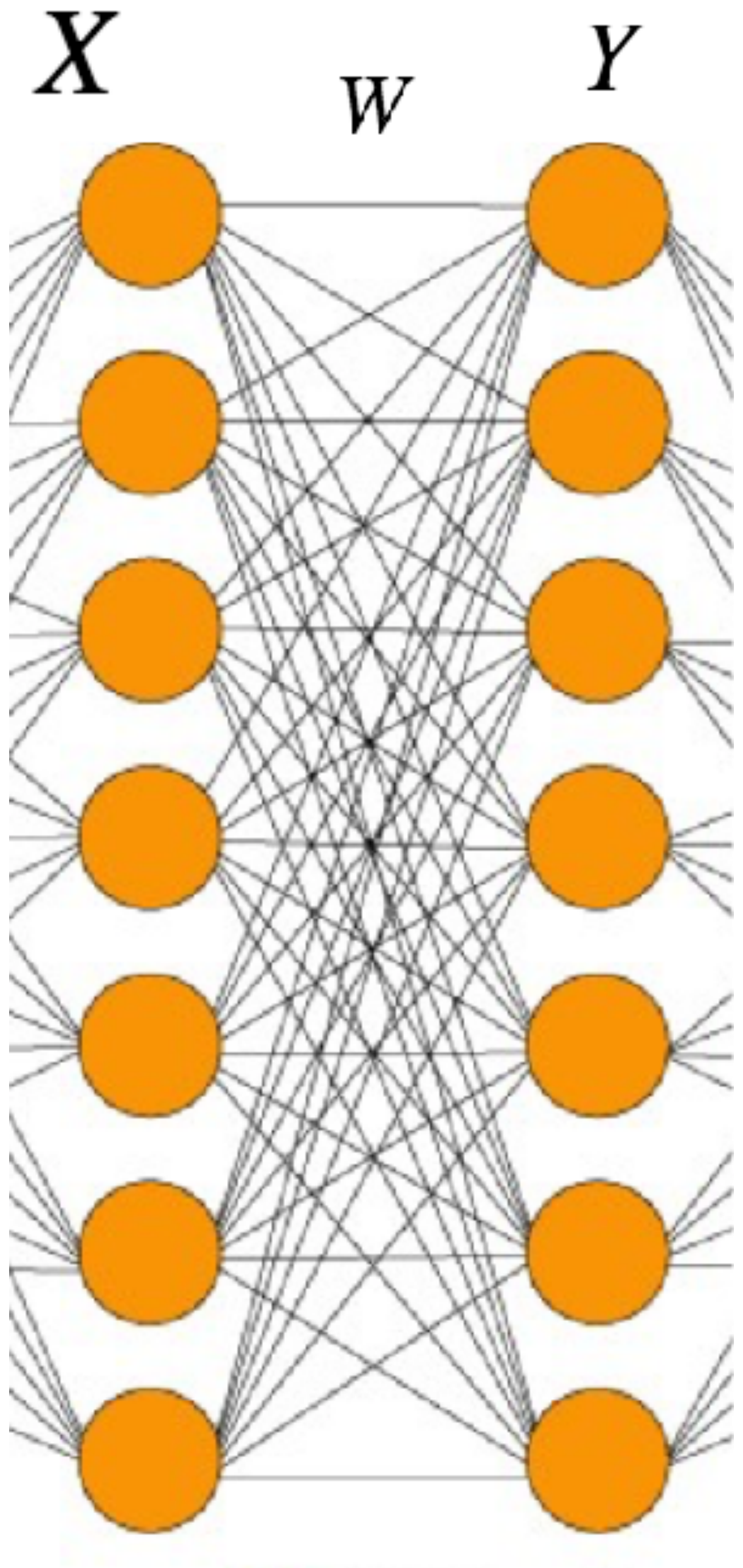
$$Y = \boxed{f(Z)}$$

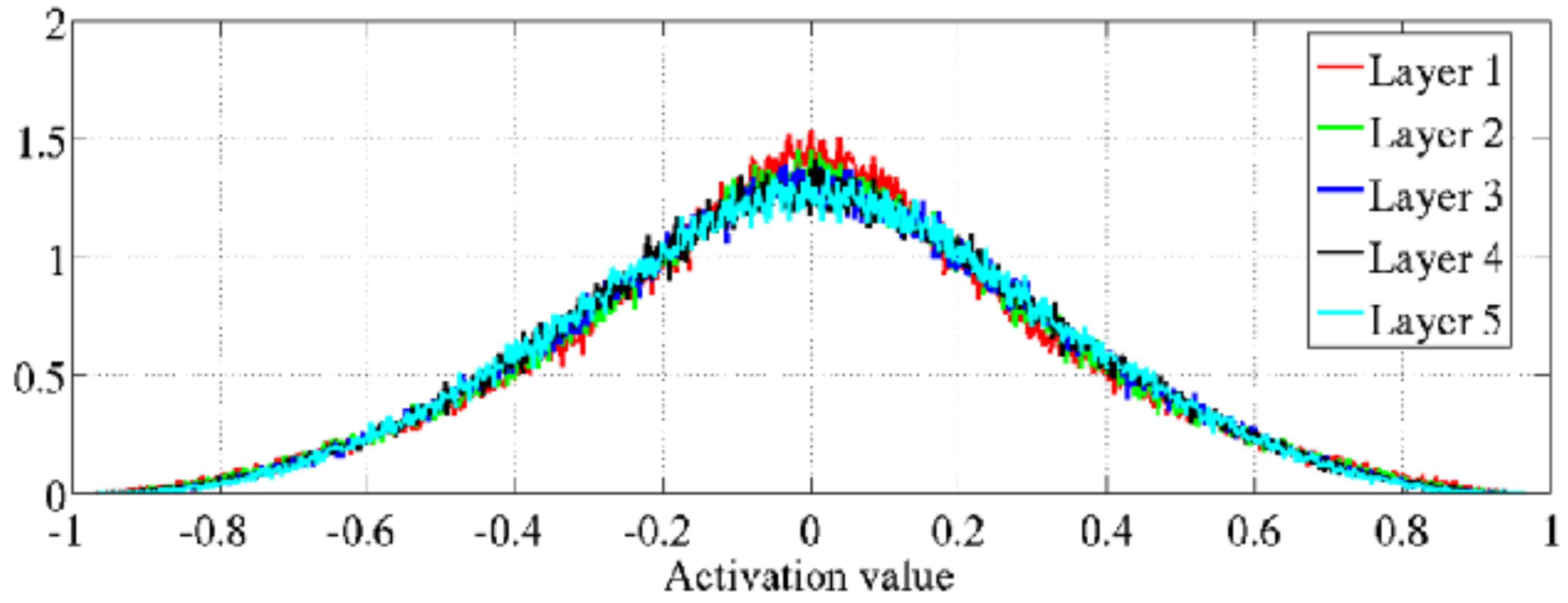
activation function

$$f(x) = x$$

$$f'(x) = 1$$







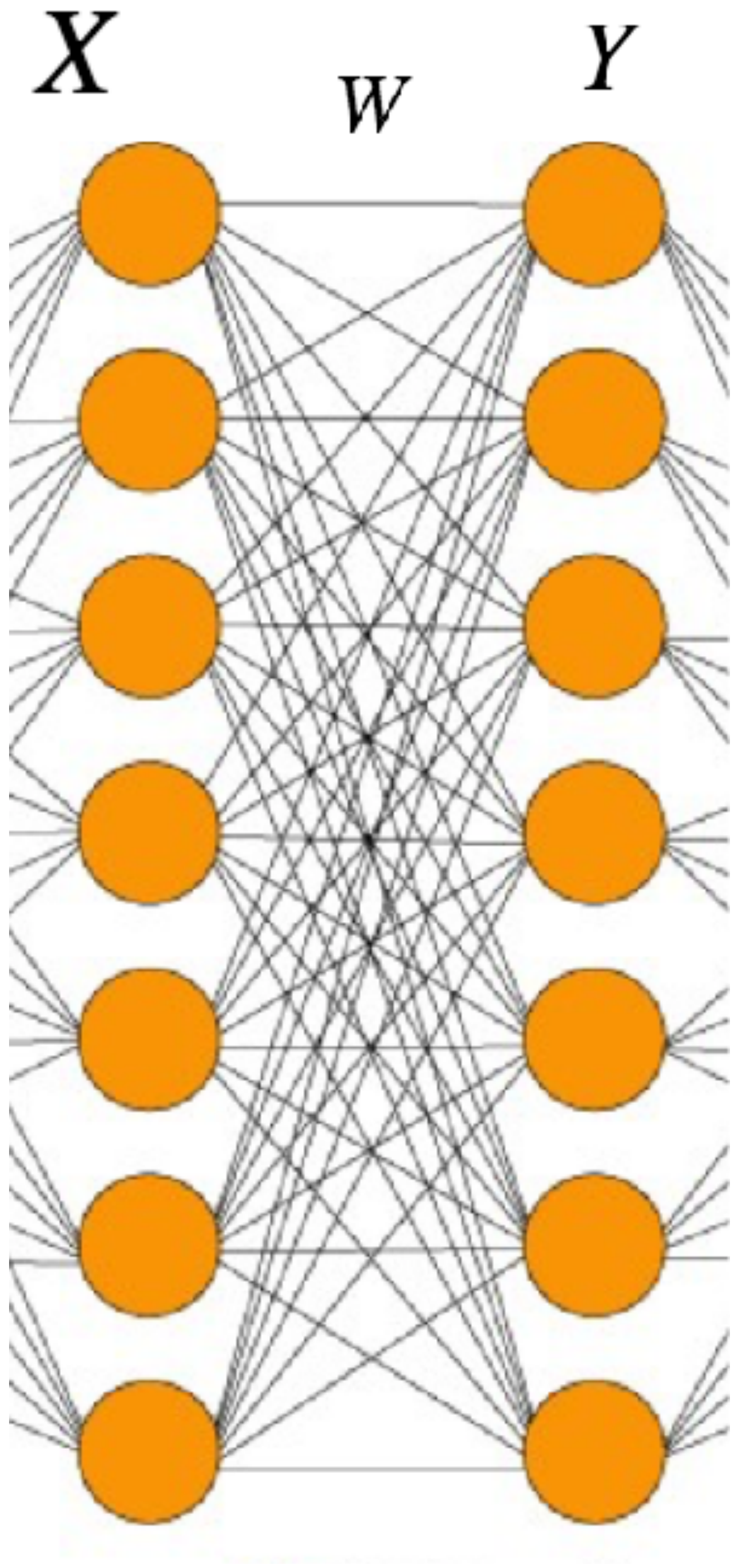
linear activation function을 사용하여
가장 보편적인 weight initialization 공식을 정한 뒤
이를 튜닝할 것

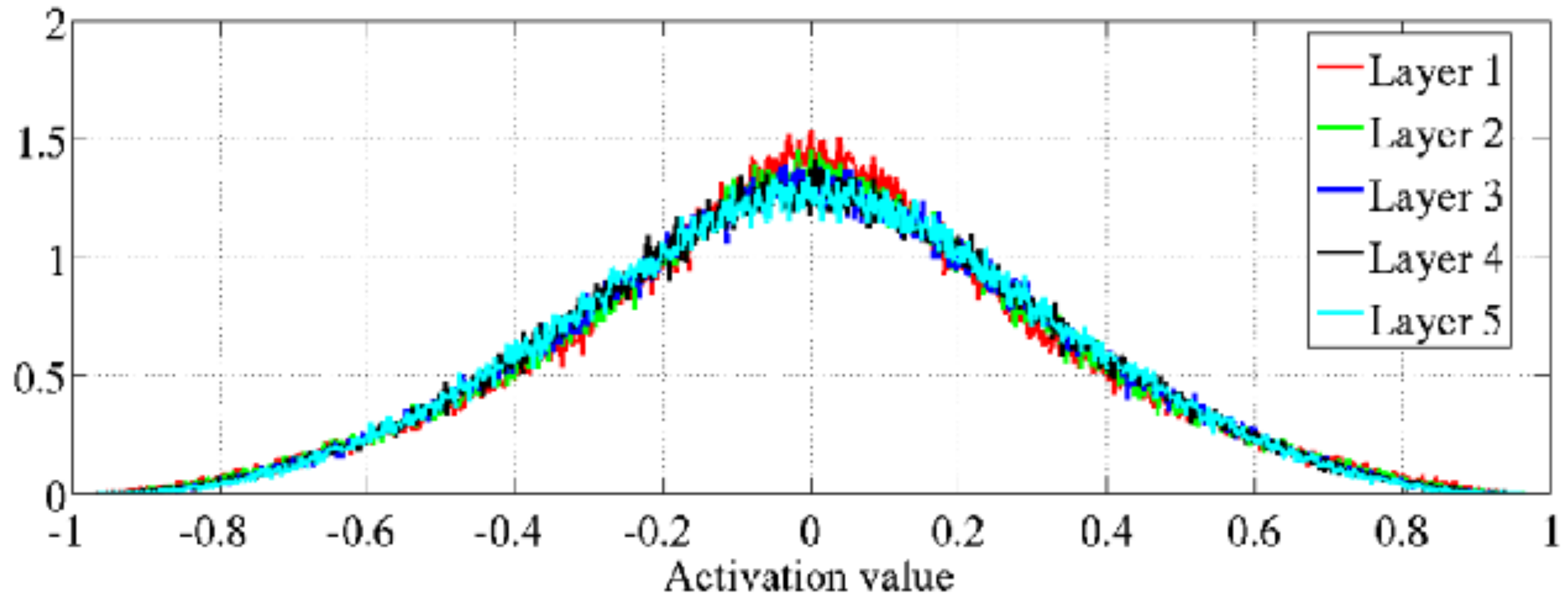
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$





Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

Suppose

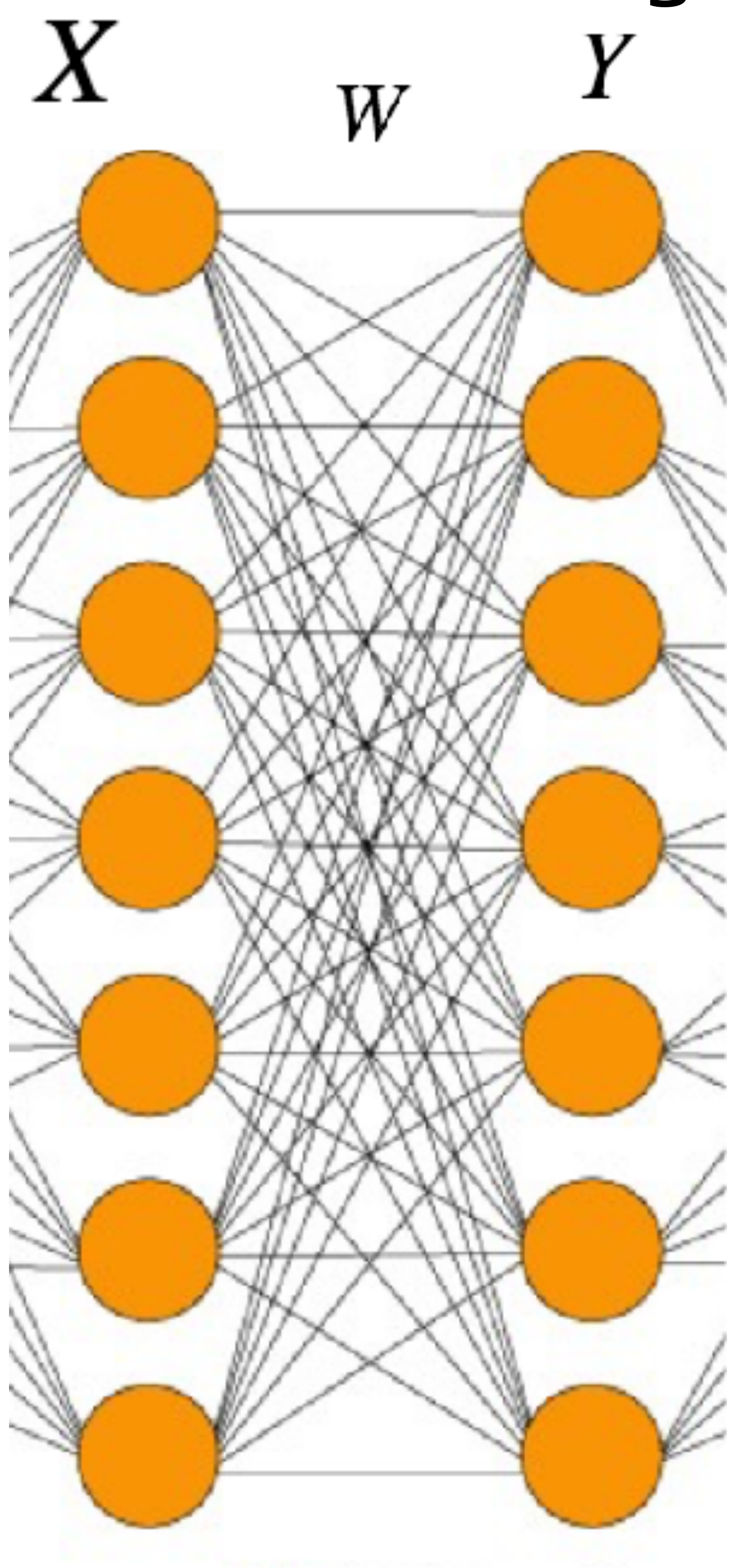
$$E(X) = 0$$
$$E(Y) = 0$$
$$E(W) = 0$$

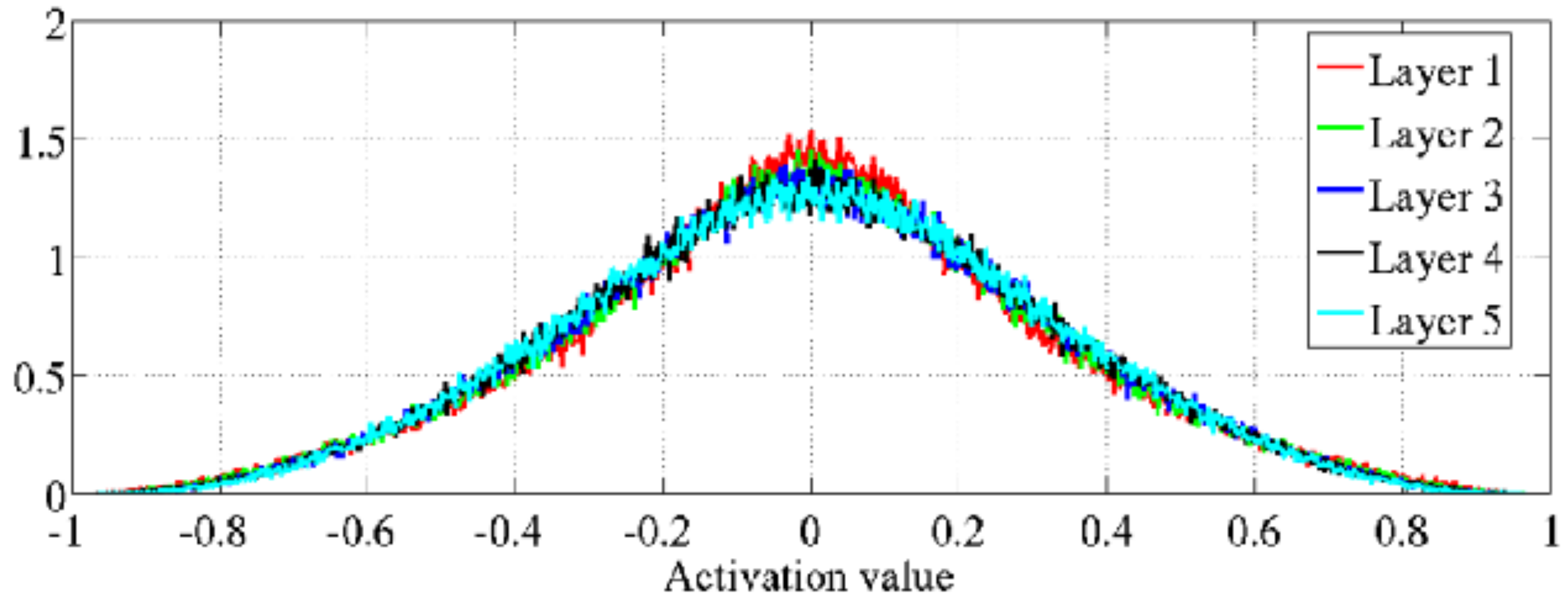
Goal

$$Var(Y) = Var(X)$$

We Want

$$Var(W) = ?$$



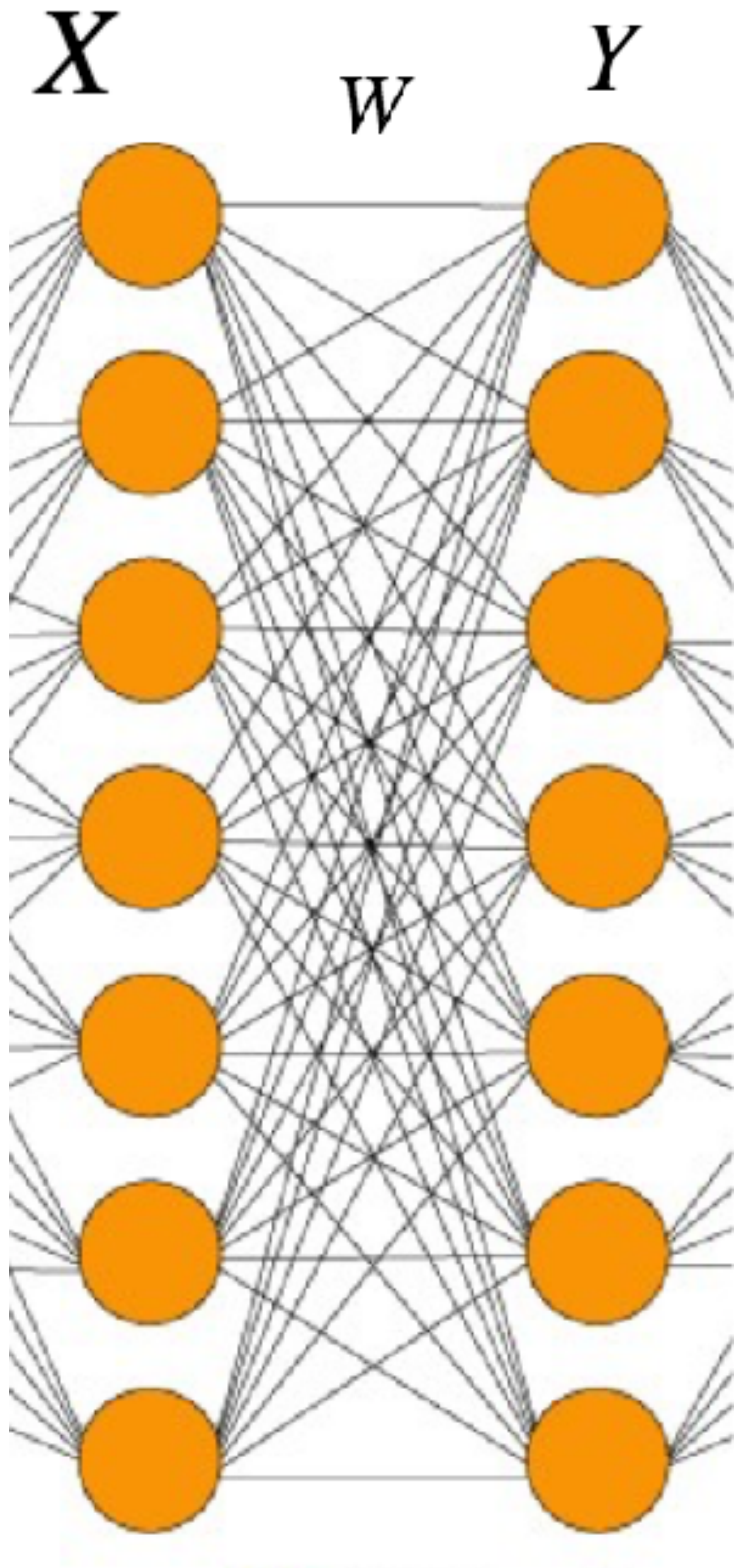


Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$



Suppose

$$E(X) = 0$$
$$E(Y) = 0$$
$$\boxed{E(W) = 0}$$

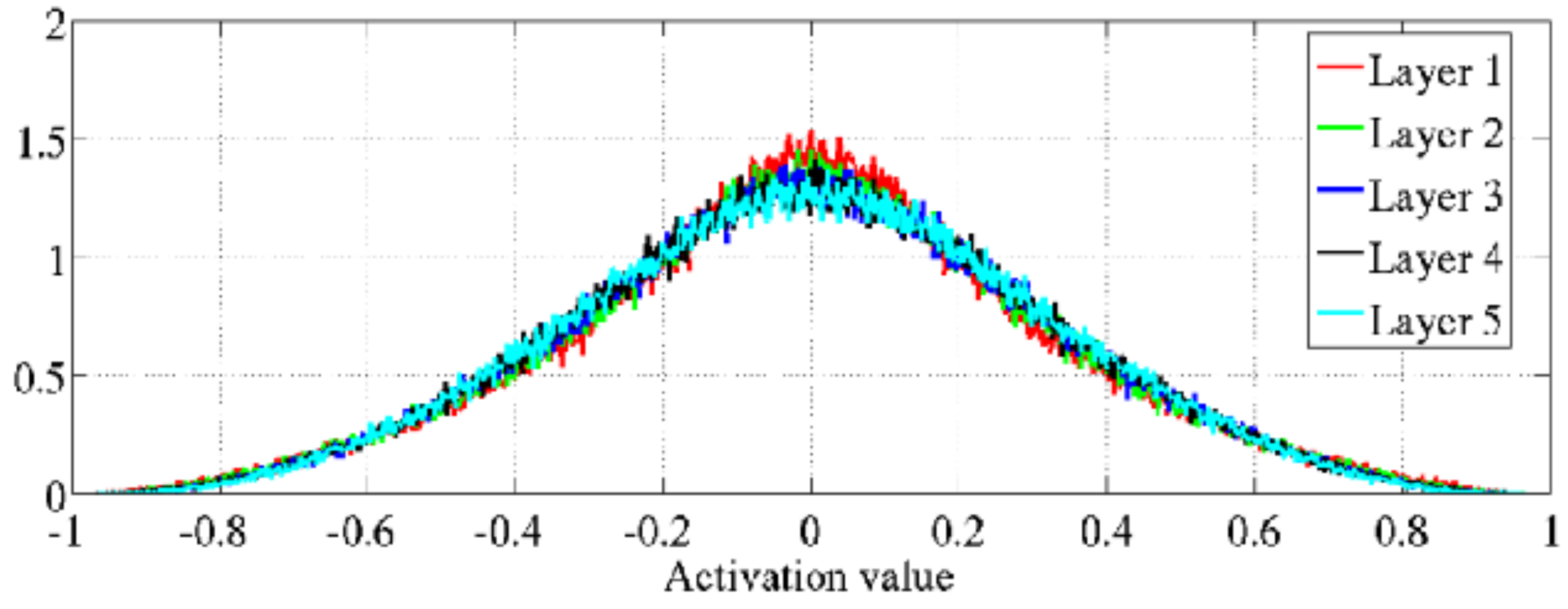
weight는 우리가 초기화하기 때문에
언제든지 E(W) = 0으로 초기화 할 수 있음

Goal

$$Var(Y) = Var(X)$$

We Want

$$Var(W) = ?$$



Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

- 주의 -
이 부분은 activation function을
어떤 것을 사용하느냐에 따라 달라짐.
즉, sigmoid나 tanh에는 적용되지만
ReLU 계열에서는 적용하지 않는다.

Suppose

$$\boxed{\begin{aligned} E(X) &= 0 \\ E(Y) &= 0 \end{aligned}}$$

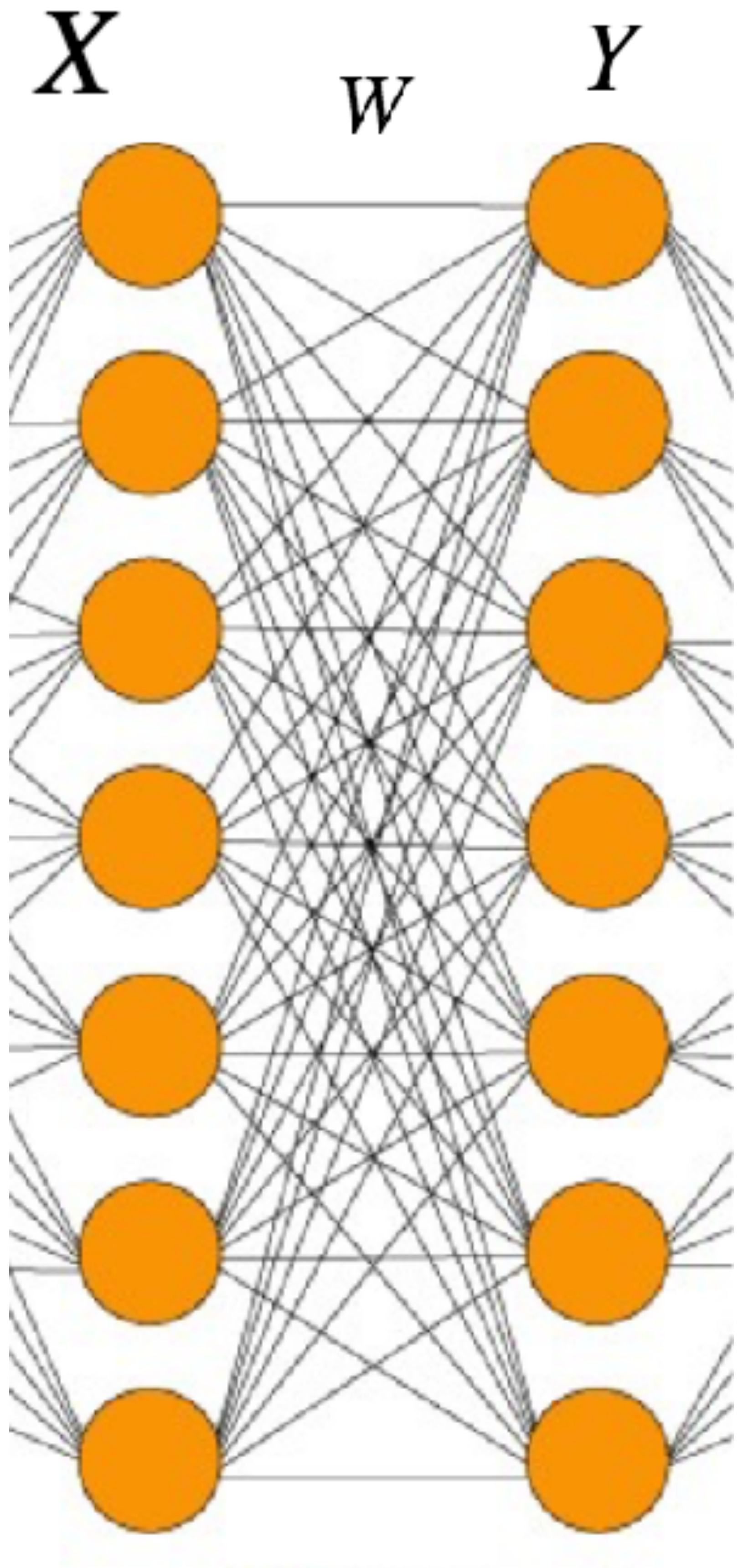
$$E(W) = 0$$

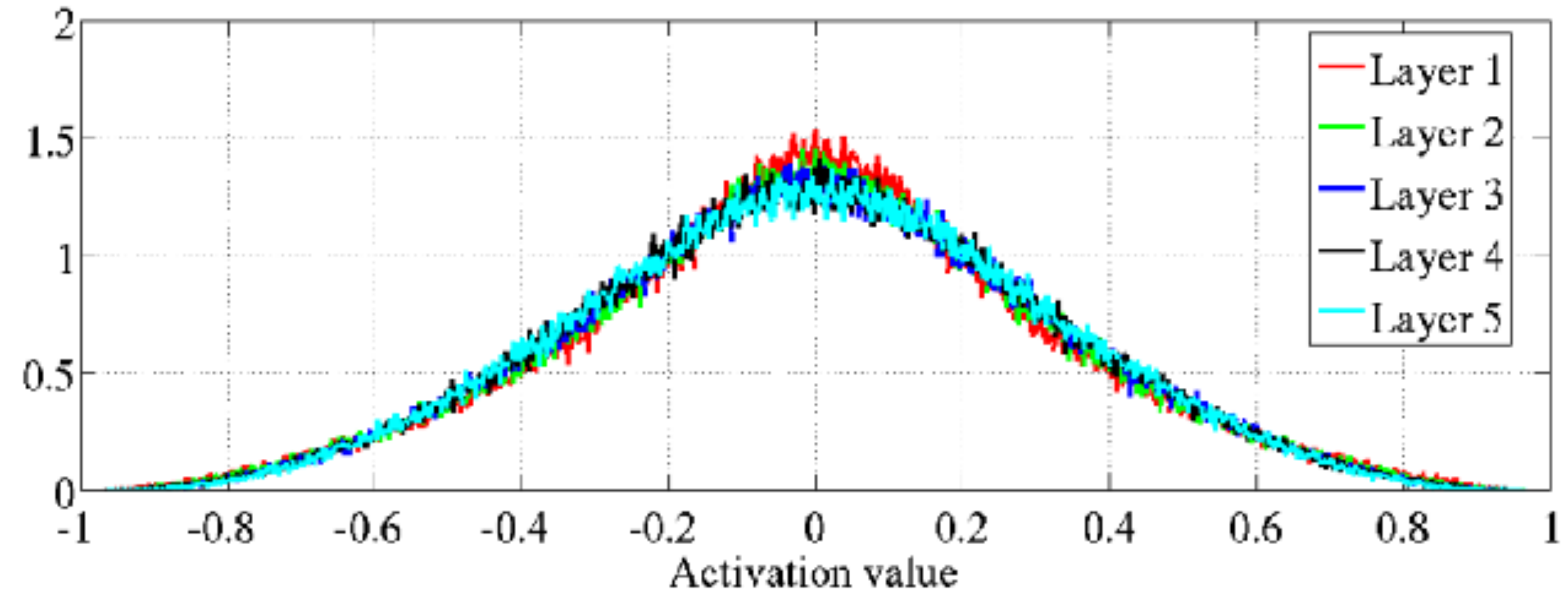
Goal

$$Var(Y) = Var(X)$$

We Want

$$Var(W) = ?$$



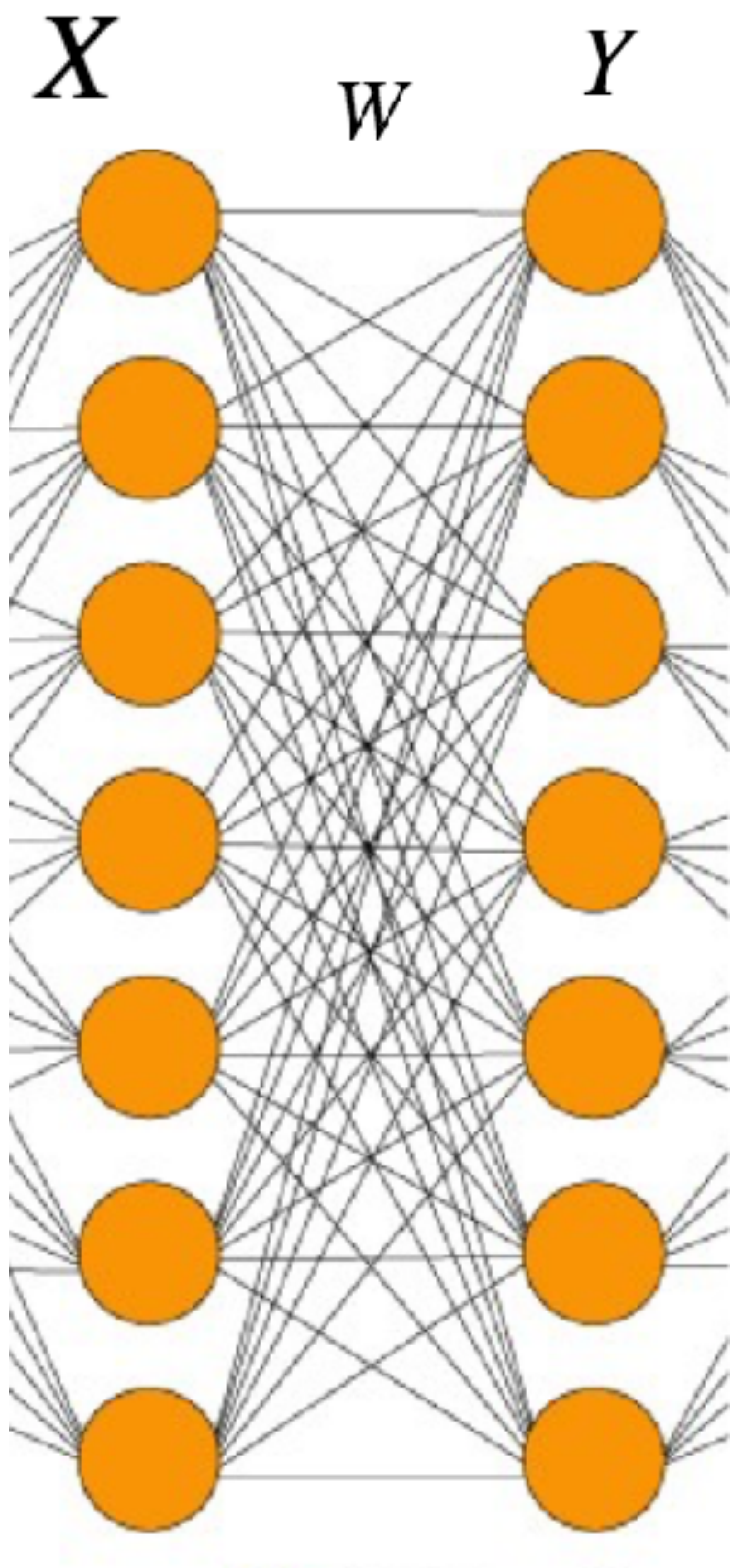


Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$



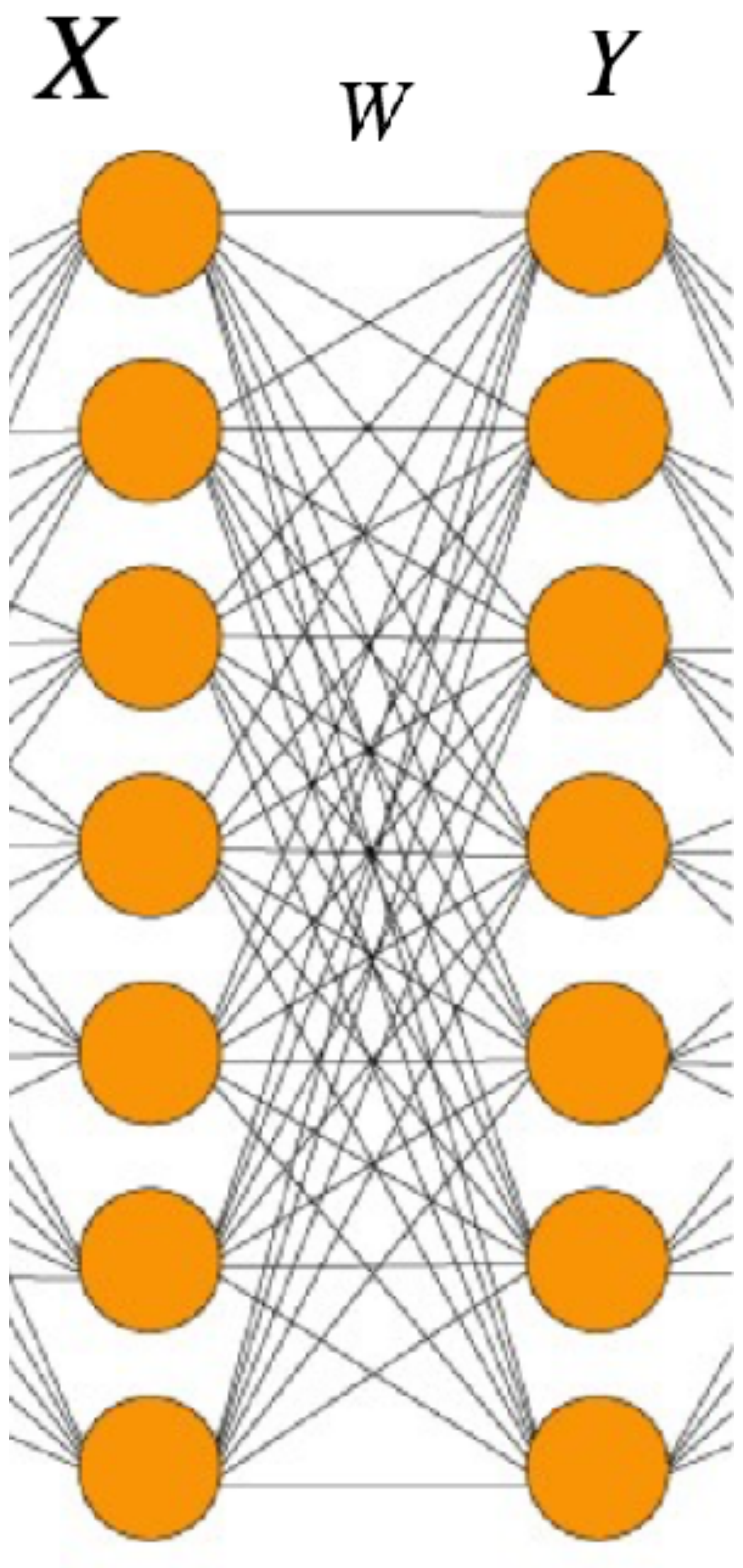
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$Var(Y) = Var(W \cdot X)$$



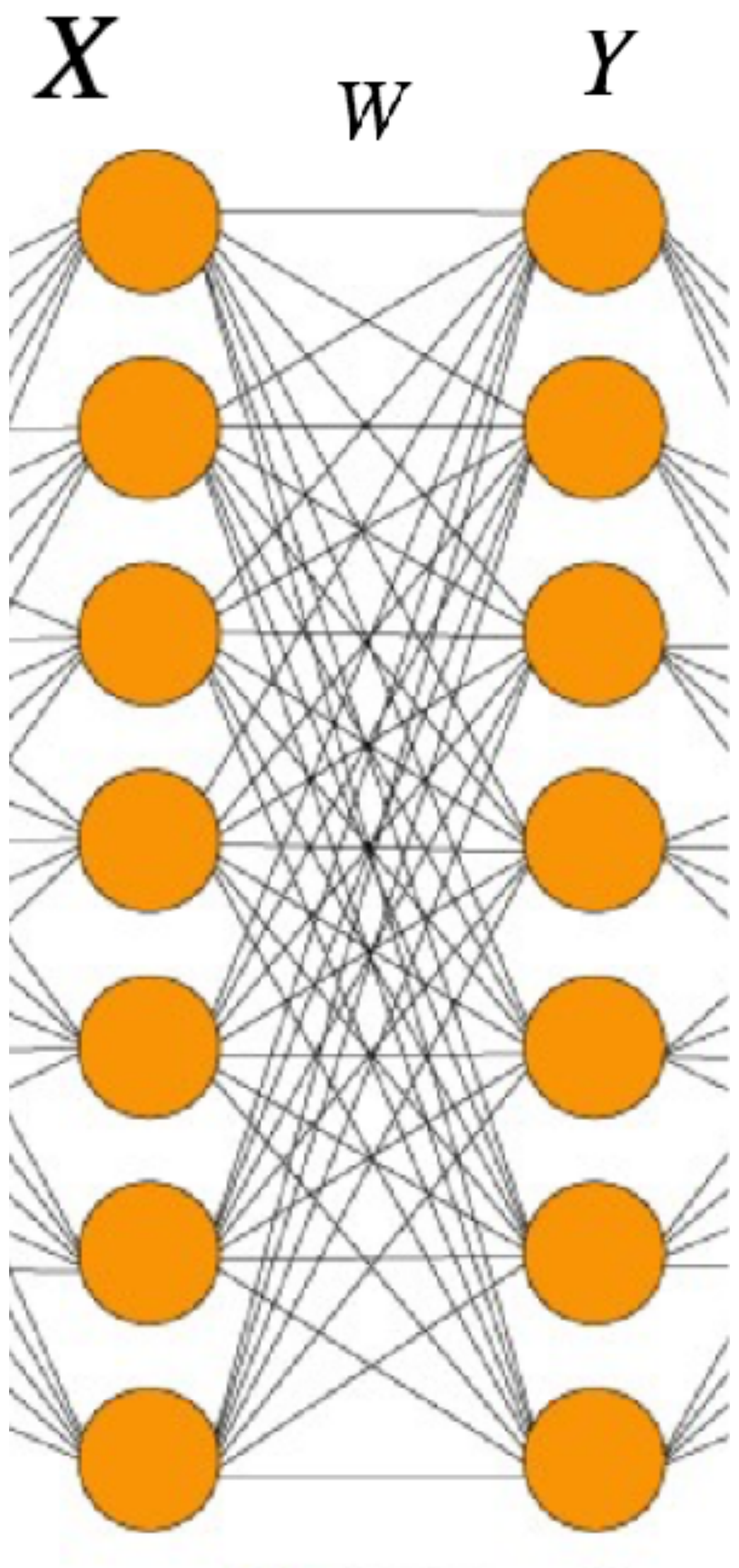
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \end{aligned}$$



n = 뉴런의 갯수

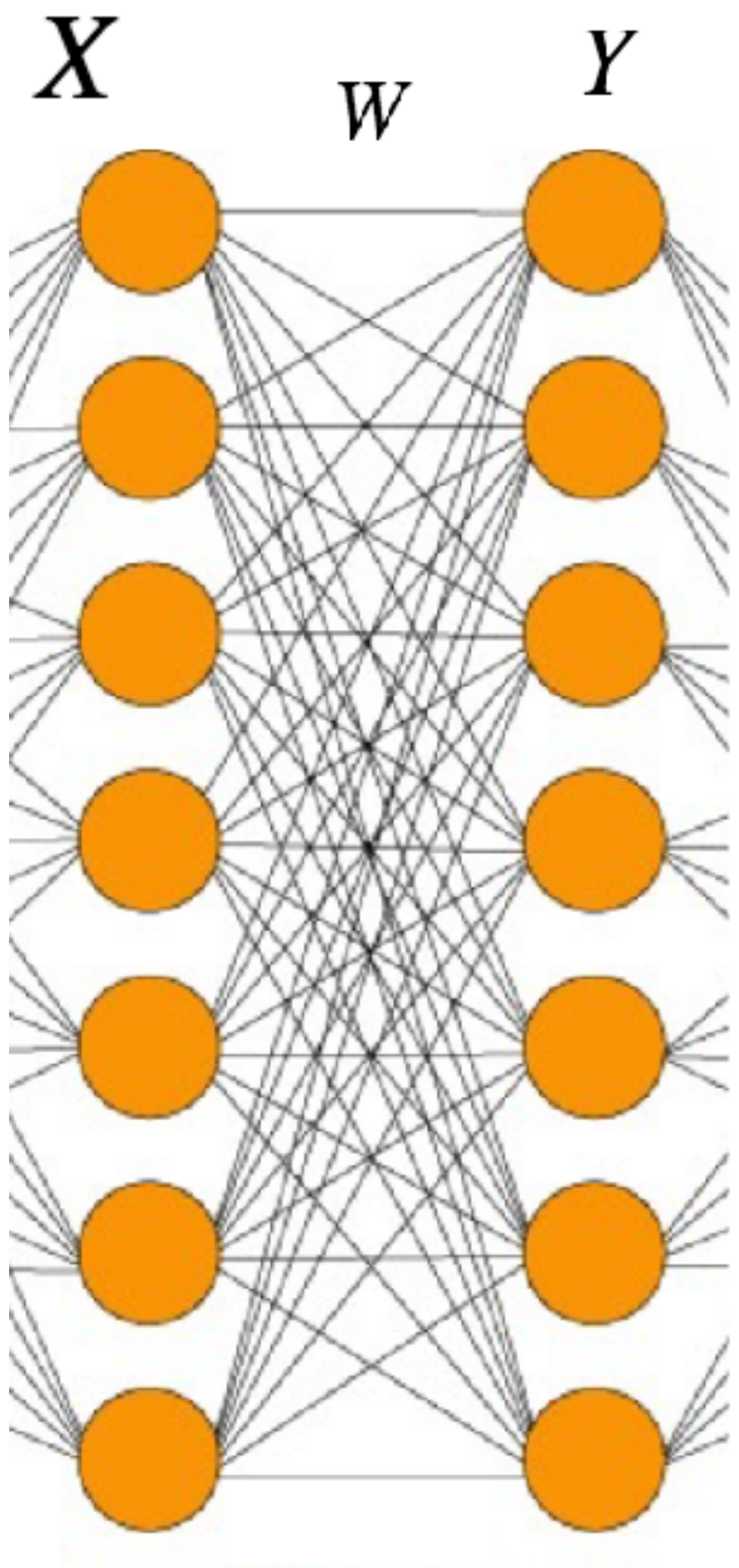
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \end{aligned}$$



$$\operatorname{Var}\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} \operatorname{Var}(X_i).$$

Sum of uncorrelated variables
See Wikipedia: https://goo.gl/Dpbw6k

n = 뉴런의 갯수

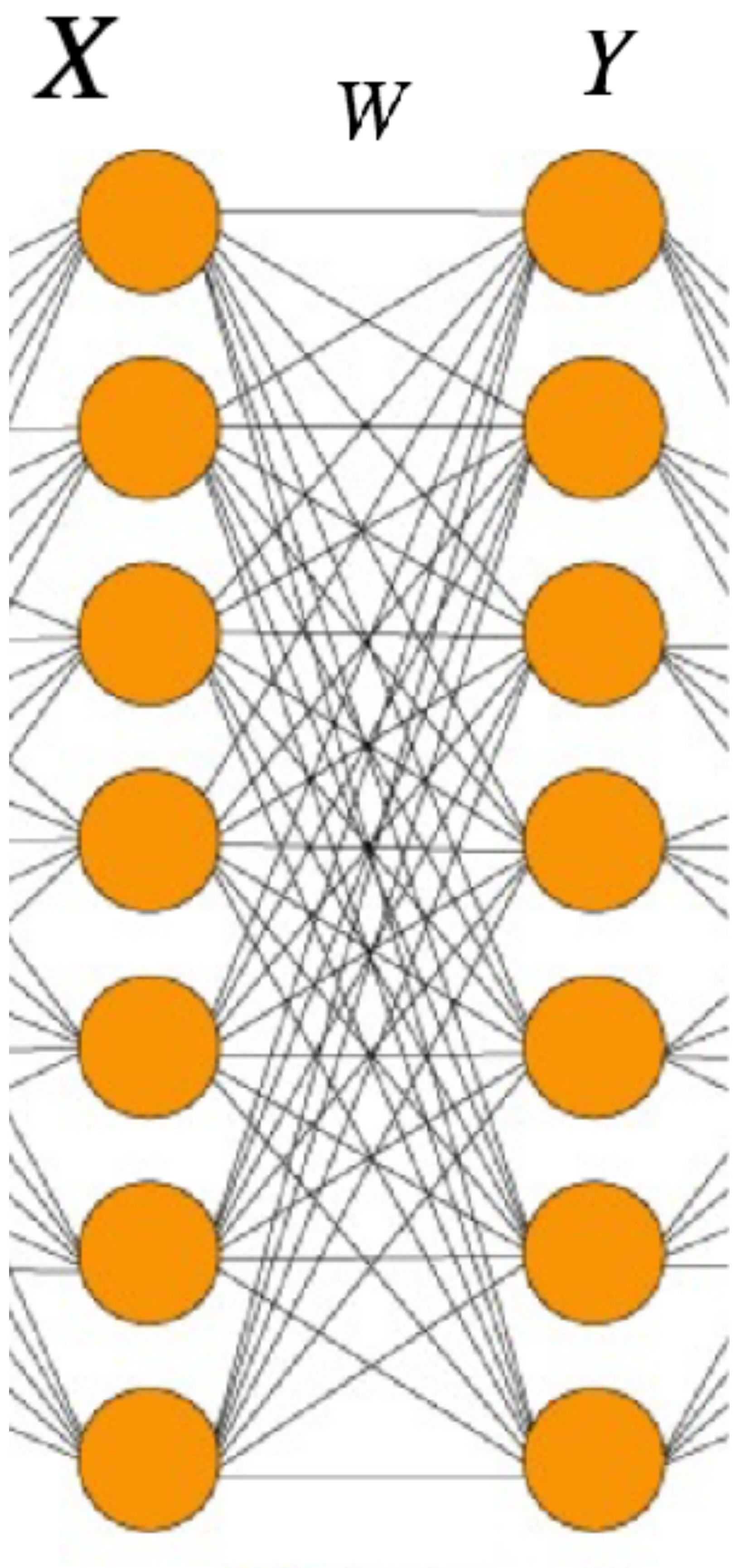
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \\ &= Var(w_1 x_1) + Var(w_2 x_2) + \ldots + Var(w_n x_n) \end{aligned}$$



n = 뉴런의 갯수

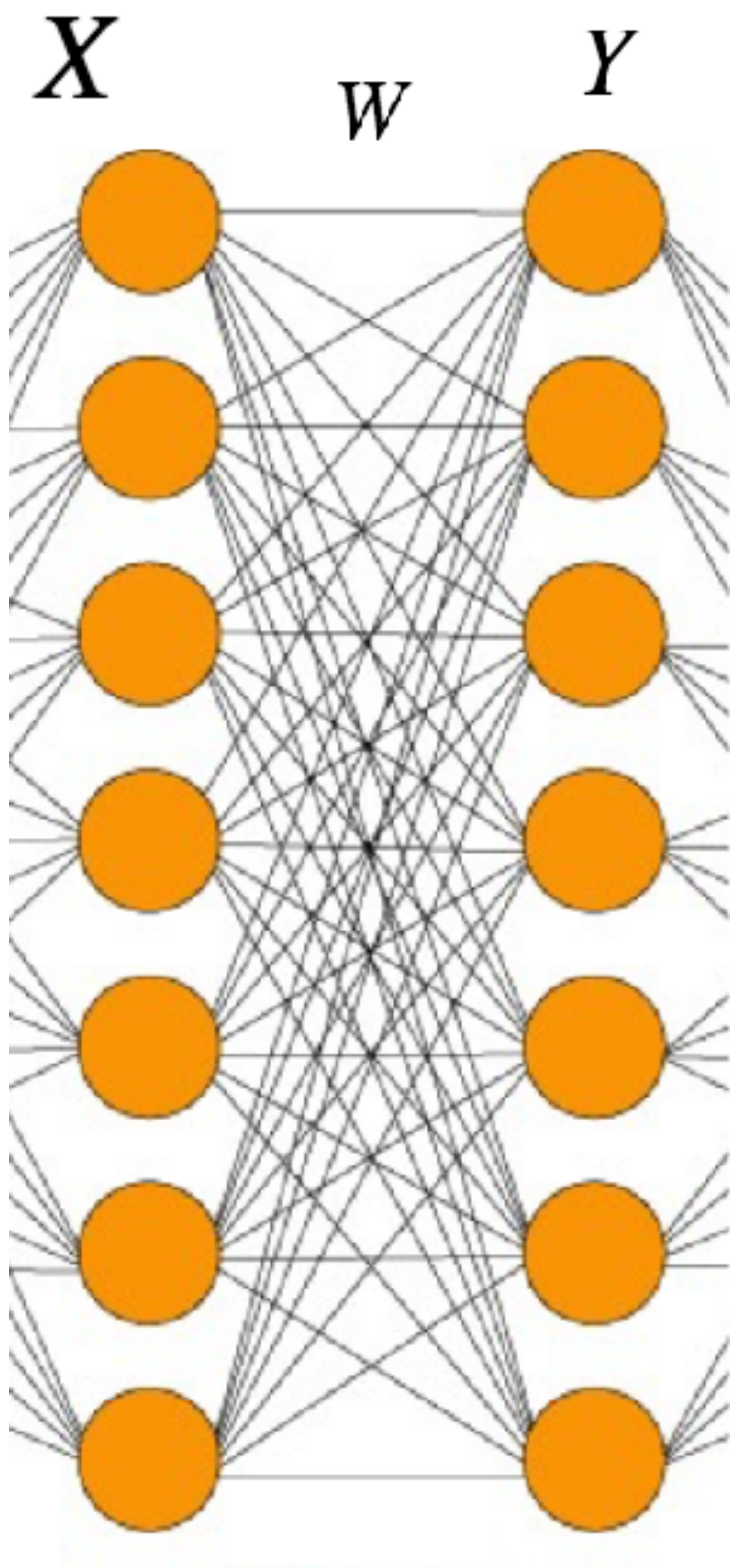
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$
\begin{aligned}
Var(Y) &= Var(W \cdot X) \\
&= Var(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \\
&= Var(w_1 x_1) + Var(w_2 x_2) + \ldots + Var(w_n x_n)
\end{aligned}
$$



$$Var(w_i x_i) = E(x_i)^2 Var(w_i) + E(w_i)^2 Var(x_i) + Var(w_i) Var(x_i)$$

Product of independent variables
See Wikipedia: https://goo.gl/bGerci

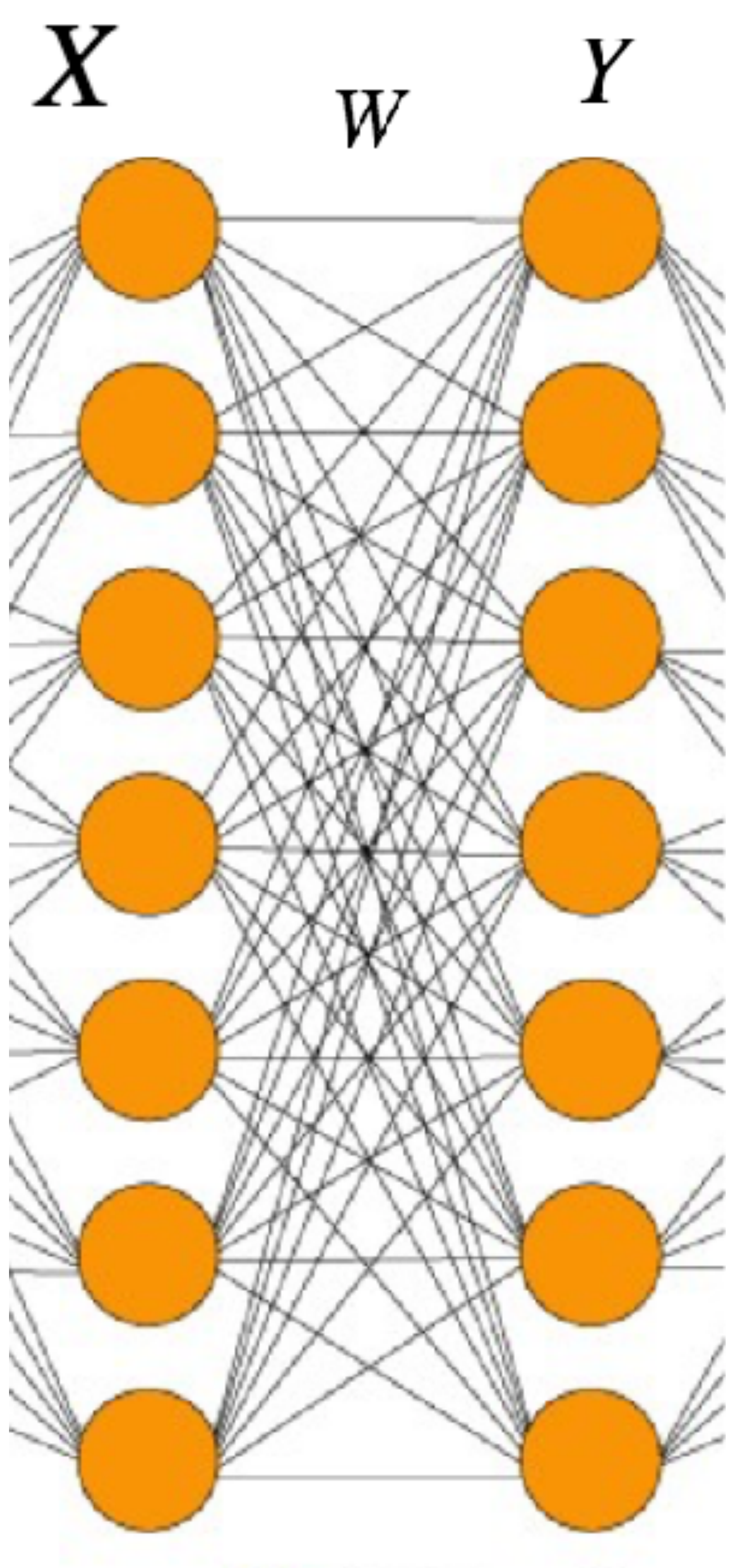
n = 뉴런의 갯수

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$



$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1x_1 + w_2x_2 + \ldots + w_nx_n) \\ &= Var(w_1x_1) + Var(w_2x_2) + \ldots + Var(w_nx_n) \end{aligned}$$

Suppose

$$E(X) = 0$$
$$E(Y) = 0$$
$$E(W) = 0$$

$$Var(w_ix_i) = E(x_i)^2 Var(w_i) + E(w_i)^2 Var(x_i) + Var(w_i)Var(x_i)$$

Product of independent variables
See Wikipedia: https://goo.gl/bGerci

n = 뉴런의 갯수

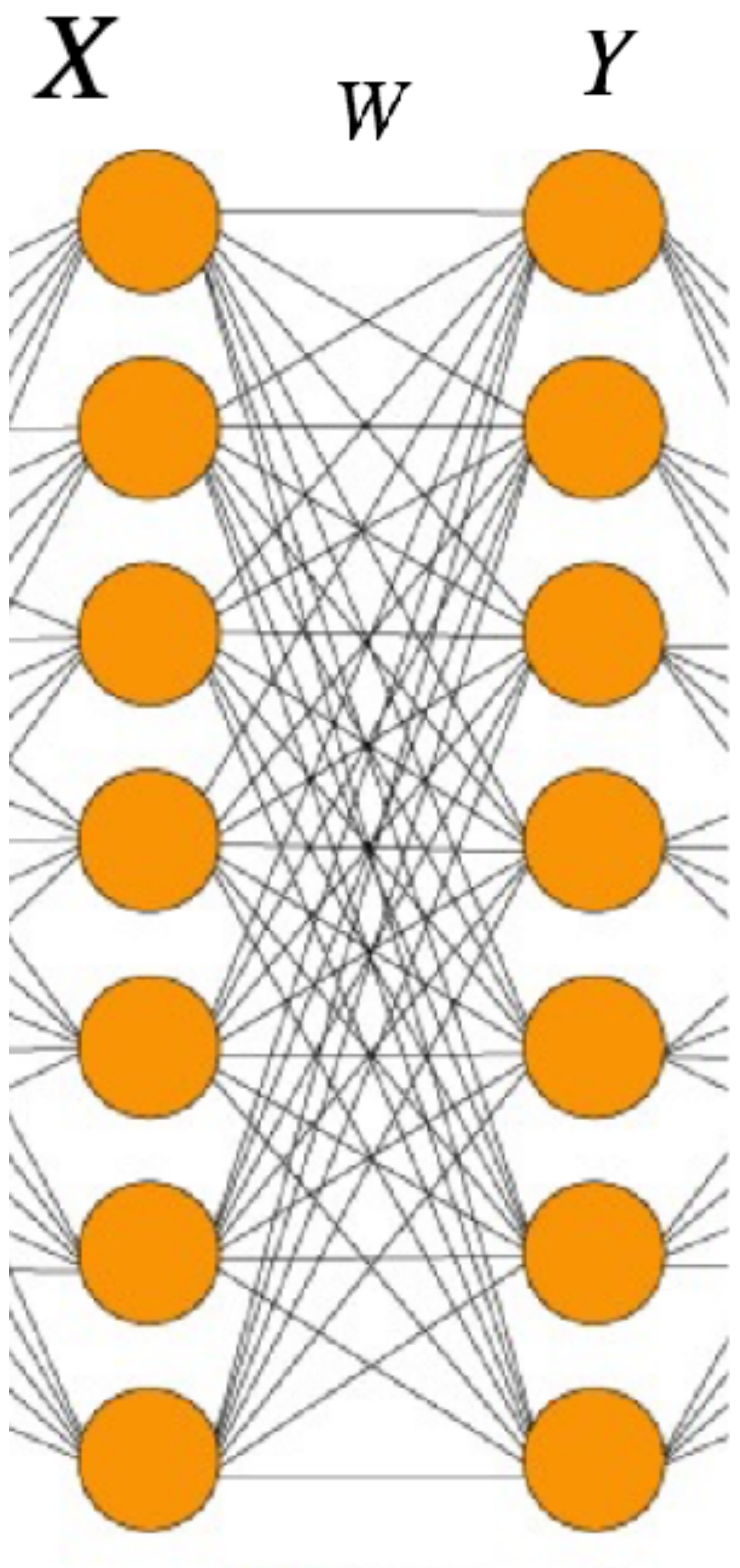
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1x_1 + w_2x_2 + \ldots + w_nx_n) \\ &= Var(w_1x_1) + Var(w_2x_2) + \ldots + Var(w_nx_n) \end{aligned}$$



Suppose

$$E(X) = 0$$
$$E(Y) = 0$$
$$E(W) = 0$$

$$\begin{aligned} Var(w_ix_i) &= E(x_i)^2Var(w_i) + E(w_i)^2Var(x_i) + Var(w_i)Var(x_i) \\ &= Var(w_i)Var(x_i) \end{aligned}$$

n = 뉴런의 갯수

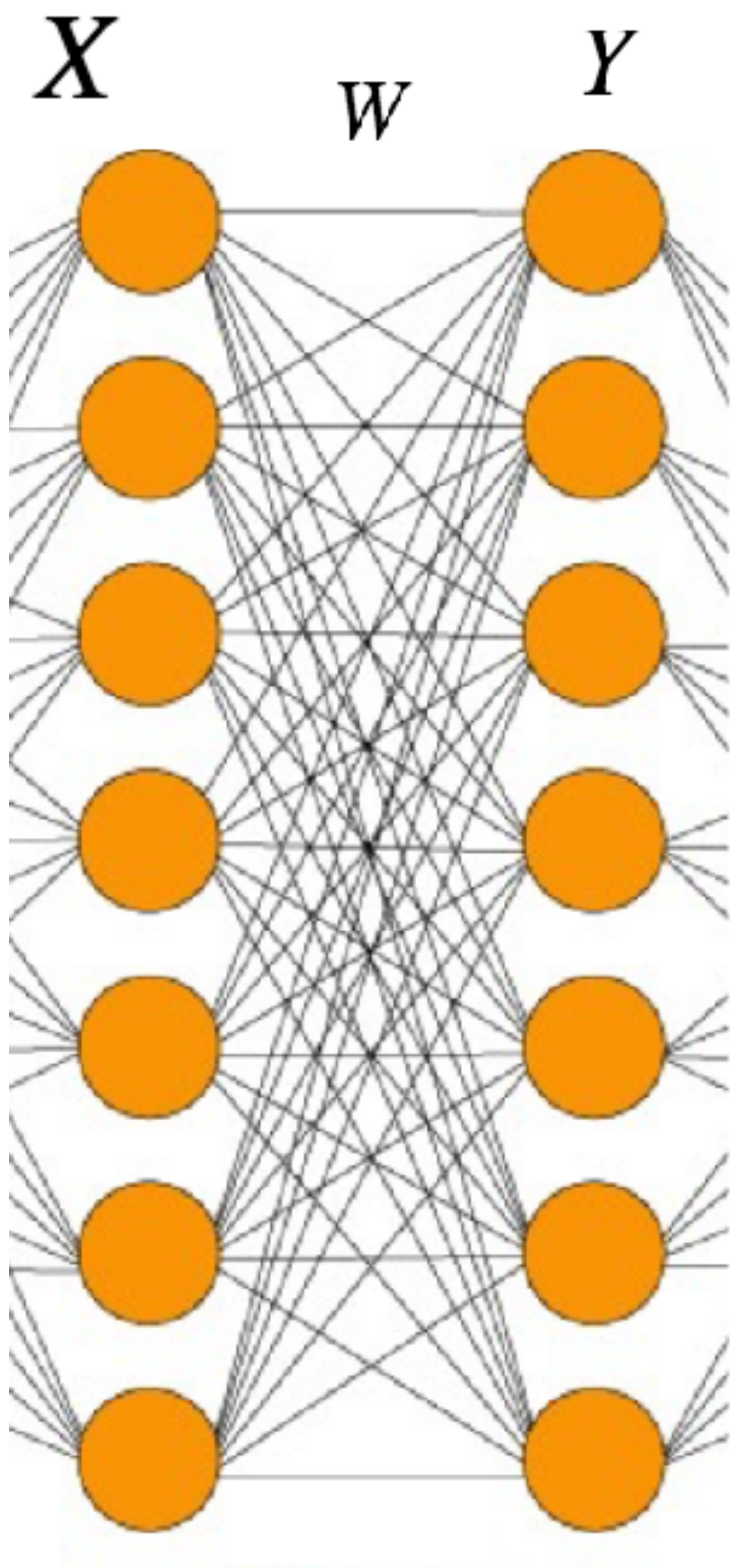
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1x_1 + w_2x_2 + \ldots + w_nx_n) \\ &= Var(w_1x_1) + Var(w_2x_2) + \ldots + Var(w_nx_n) \\ &= Var(w_1)Var(x_1) + Var(w_2)Var(x_2) + \ldots + Var(w_n)Var(x_n) \end{aligned}$$



n = 뉴런의 갯수

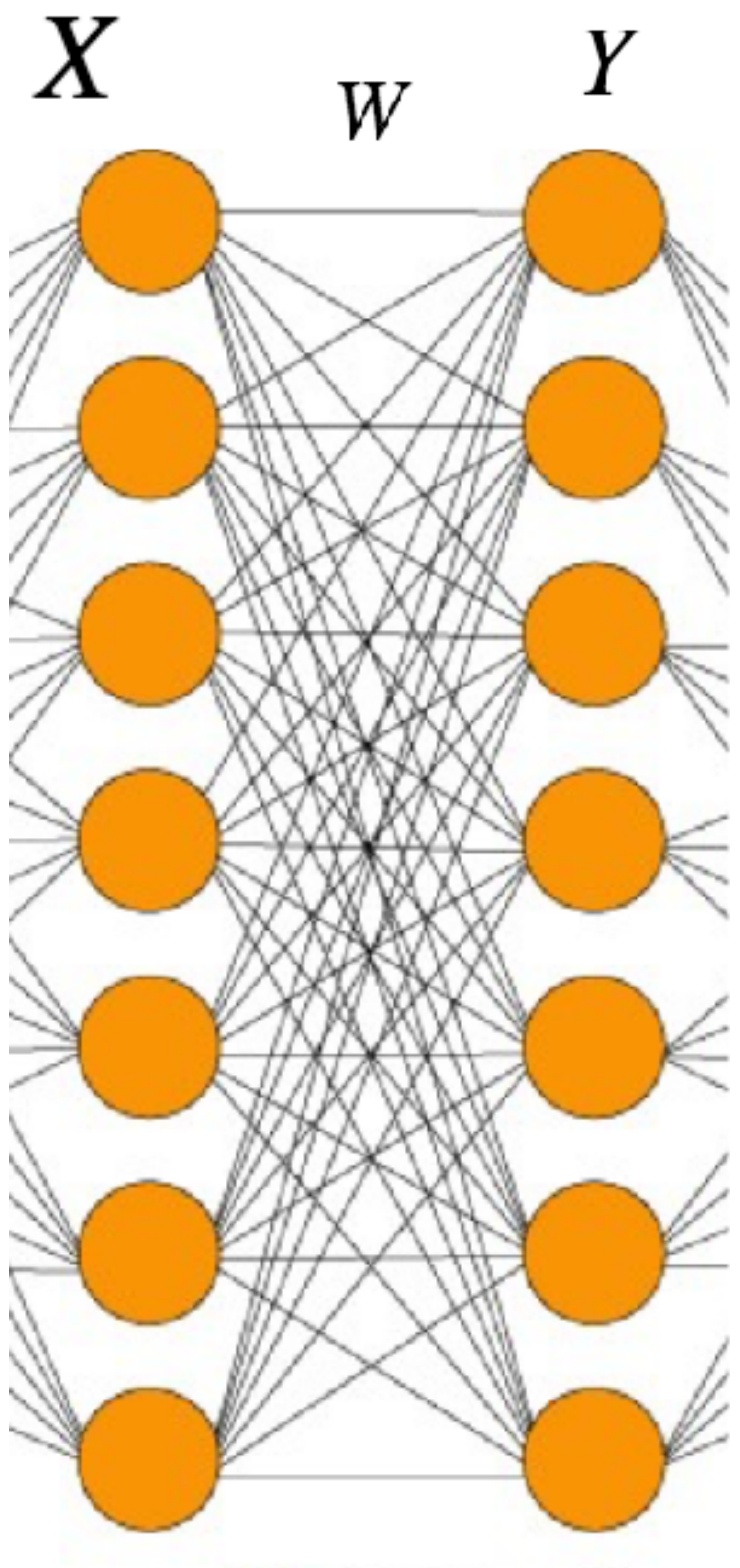
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n) \\ &= Var(w_1 x_1) + Var(w_2 x_2) + \ldots + Var(w_n x_n) \\ &= Var(w_1)Var(x_1) + Var(w_2)Var(x_2) + \ldots + Var(w_n)Var(x_n) \\ &= nVar(w)Var(x) \end{aligned}$$



n = 뉴런의 갯수

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

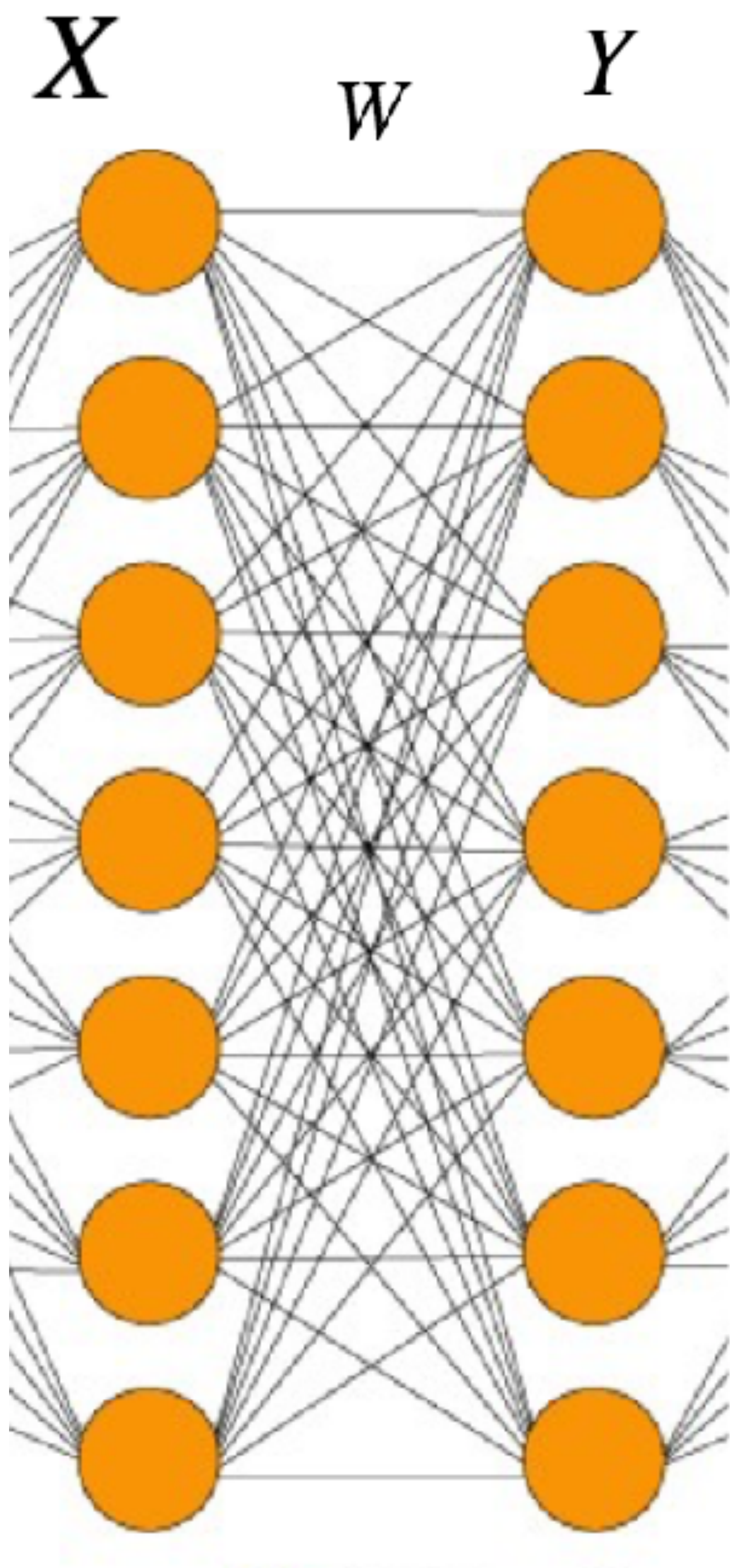
What we want

레이어가 늘어나더라도 activation value의 평균과 분산이 일정하기를 원한다.
그렇다면 이론적으로 무제한의 hidden layer를 쌓을 수 있다.

$$Y = W \cdot X$$

$$\begin{aligned} Var(Y) &= Var(W \cdot X) \\ &= Var(w_1x_1 + w_2x_2 + \ldots + w_nx_n) \\ &= Var(w_1x_1) + Var(w_2x_2) + \ldots + Var(w_nx_n) \\ &= Var(w_1)Var(x_1) + Var(w_2)Var(x_2) + \ldots + Var(w_n)Var(x_n) \\ &= nVar(w)Var(x) \end{aligned}$$

$$Var(W) = \frac{1}{n}$$



n = 뉴런의 갯수

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

실제로는 forward 연산을 할 때의 variance와 backward 연산을 할 때 variance를 모두 고려해야 한다.
그러므로 두 경우의 조화평균으로 weight의 variance를 초기화한다.

forward propagation

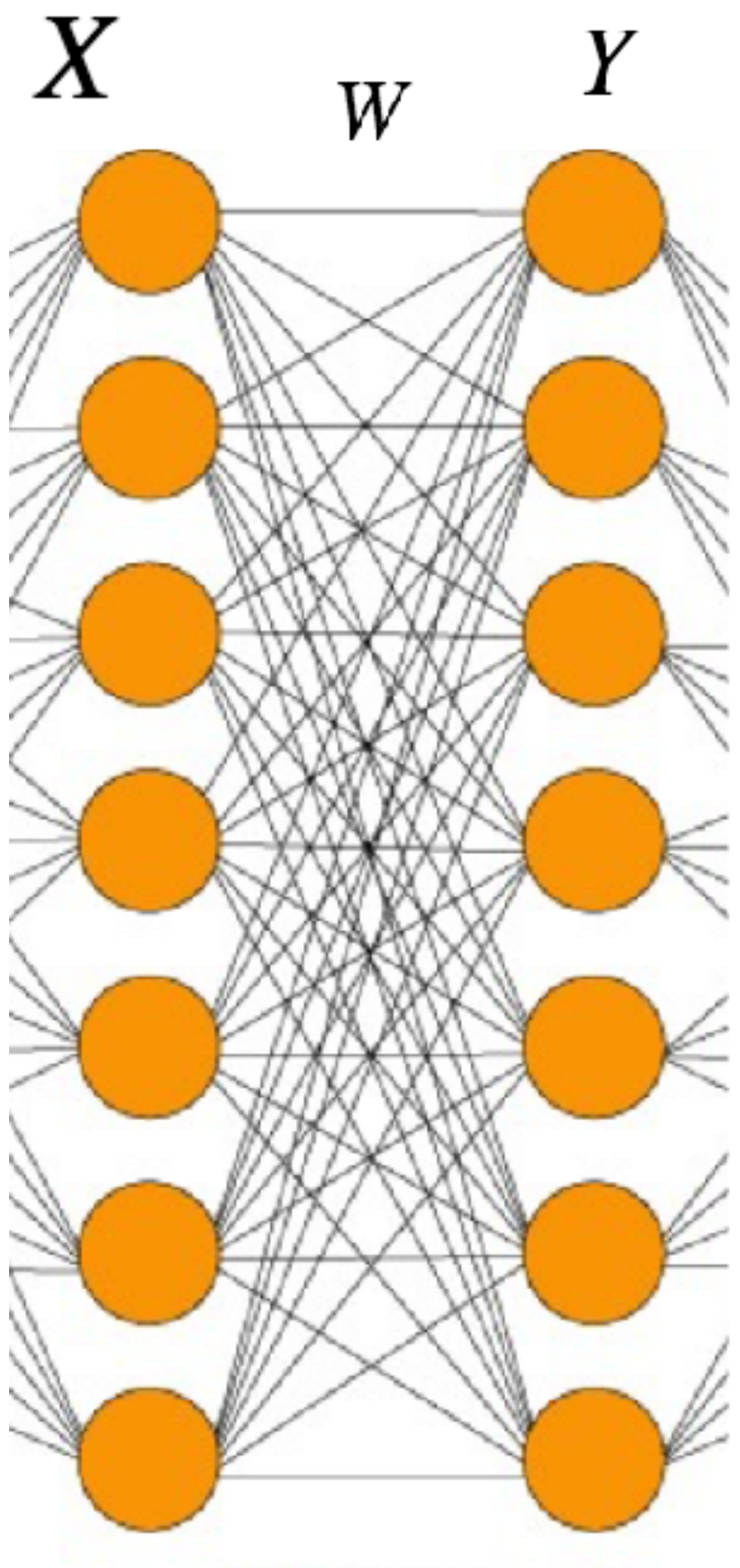
$$Var(W) = \frac{1}{n_{in}}$$

backward propagation

$$Var(W) = \frac{1}{n_{out}}$$

We Want

$$Var(W) = \frac{2}{n_{in} + n_{out}}$$



n(in) =
input 뉴런의 갯수

n(out) =
output 뉴런의 갯수

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

What we want

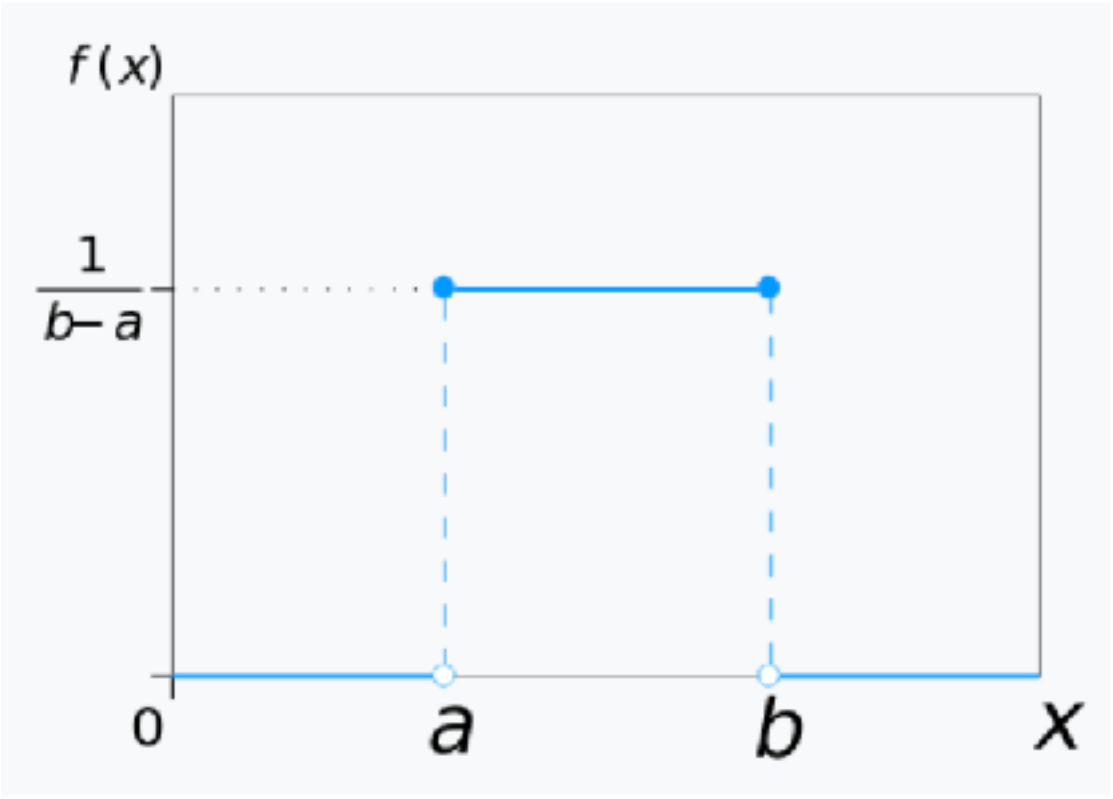
weight를 Gaussian으로 초기화할 경우와 Uniform으로 초기화할 경우 모두를 고려해야 한다

Gaussian

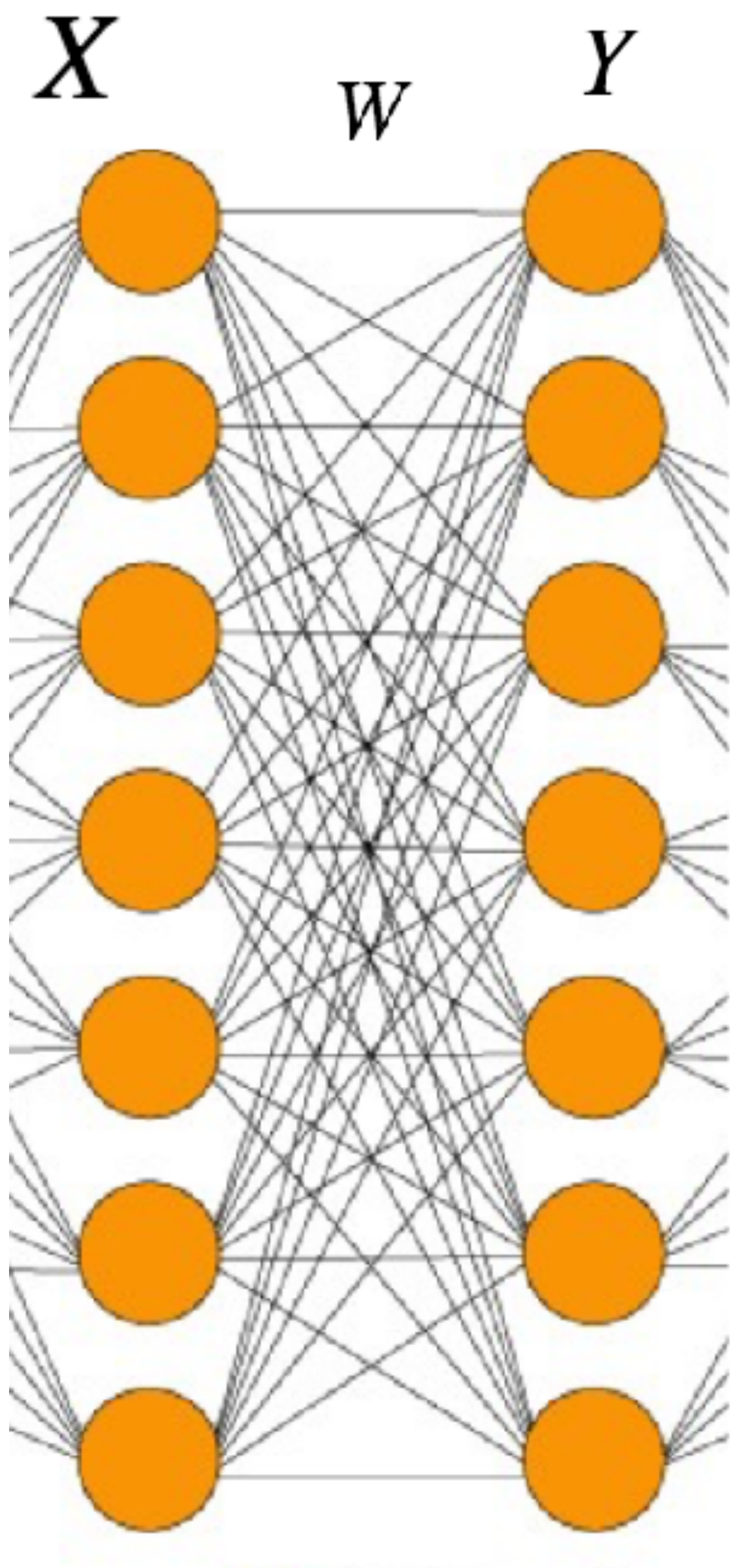
$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

Uniform

$$Var(W) = \sqrt{\frac{6}{n_{in} + n_{out}}}$$



Variance	$\frac{1}{12}(b-a)^2$



n(in) = input 뉴런의 갯수

n(out) = output 뉴런의 갯수

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

Solution 2 - normalized initialization(xavior initialization)

그러므로, 그동안 경험적(heuristic)인 방식으로 초기화를 하는 것을 넘어서서
더 정확하고 보편적으로 쓰일 수 있는 weight initialization을 찾았다고 볼 수 있다

$$W_{ij} \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right],$$

before

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right]$$

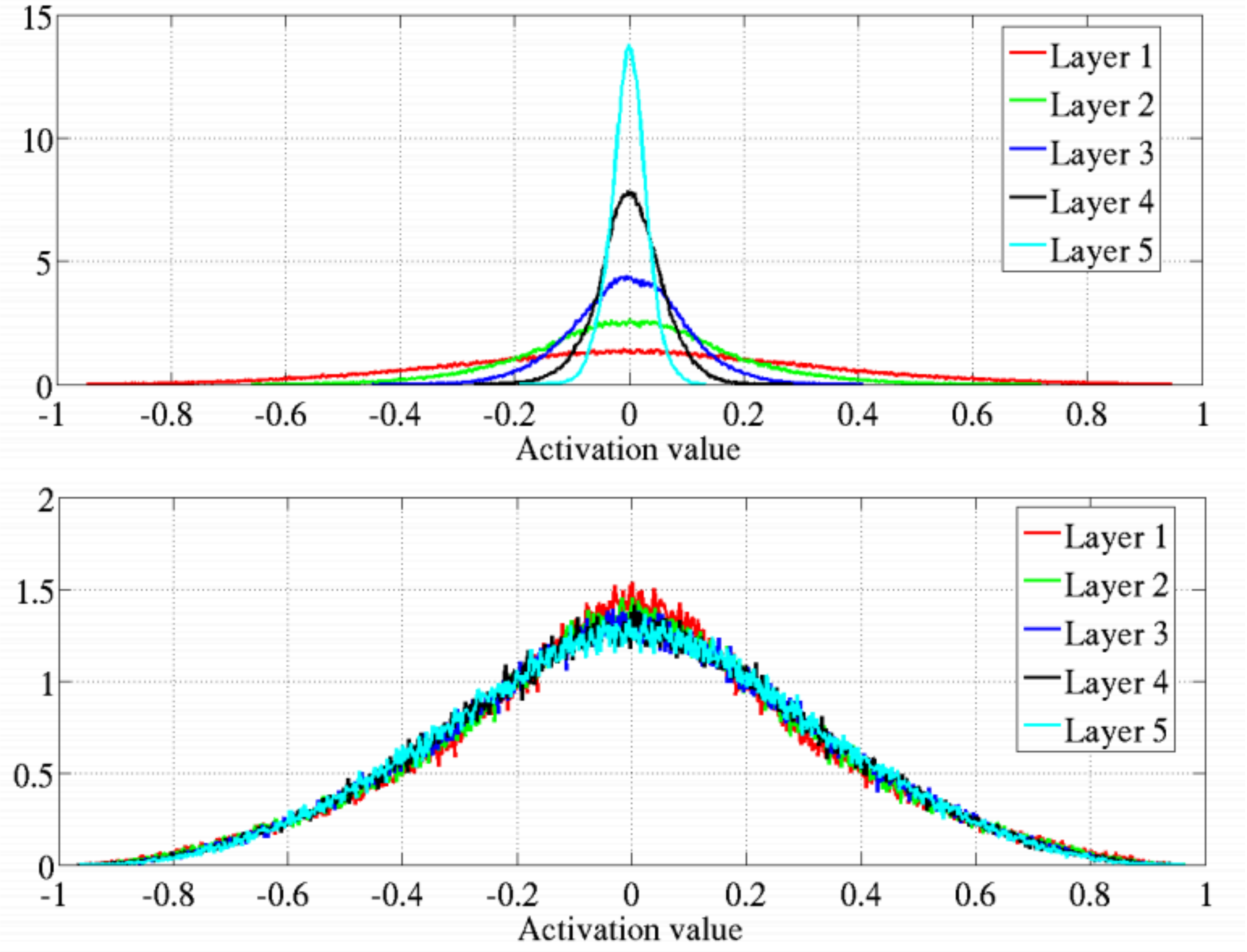
after

```python
a = np.sqrt(np.sqrt(6.0 / 64 + 100))

w = np.random.uniform(low=-a, high=+a, size=(64, 100))
```

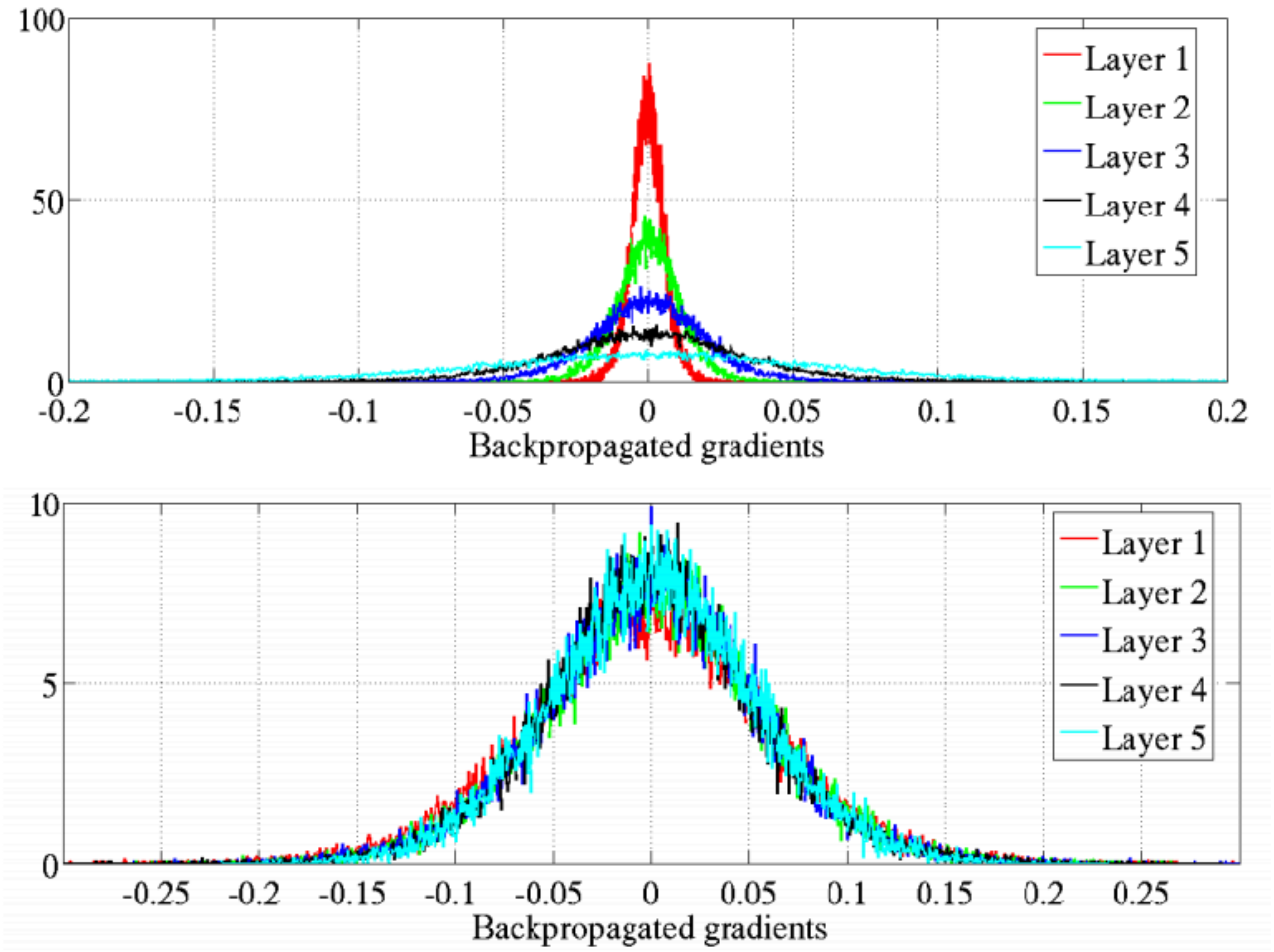
Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

Solution 2 - normalized initialization(xavior initialization)

activation(forward), gradient(backward) 모두 이전에 비해 고르게 분포되어 있는 것을 확인할 수 있다.



forward propagation

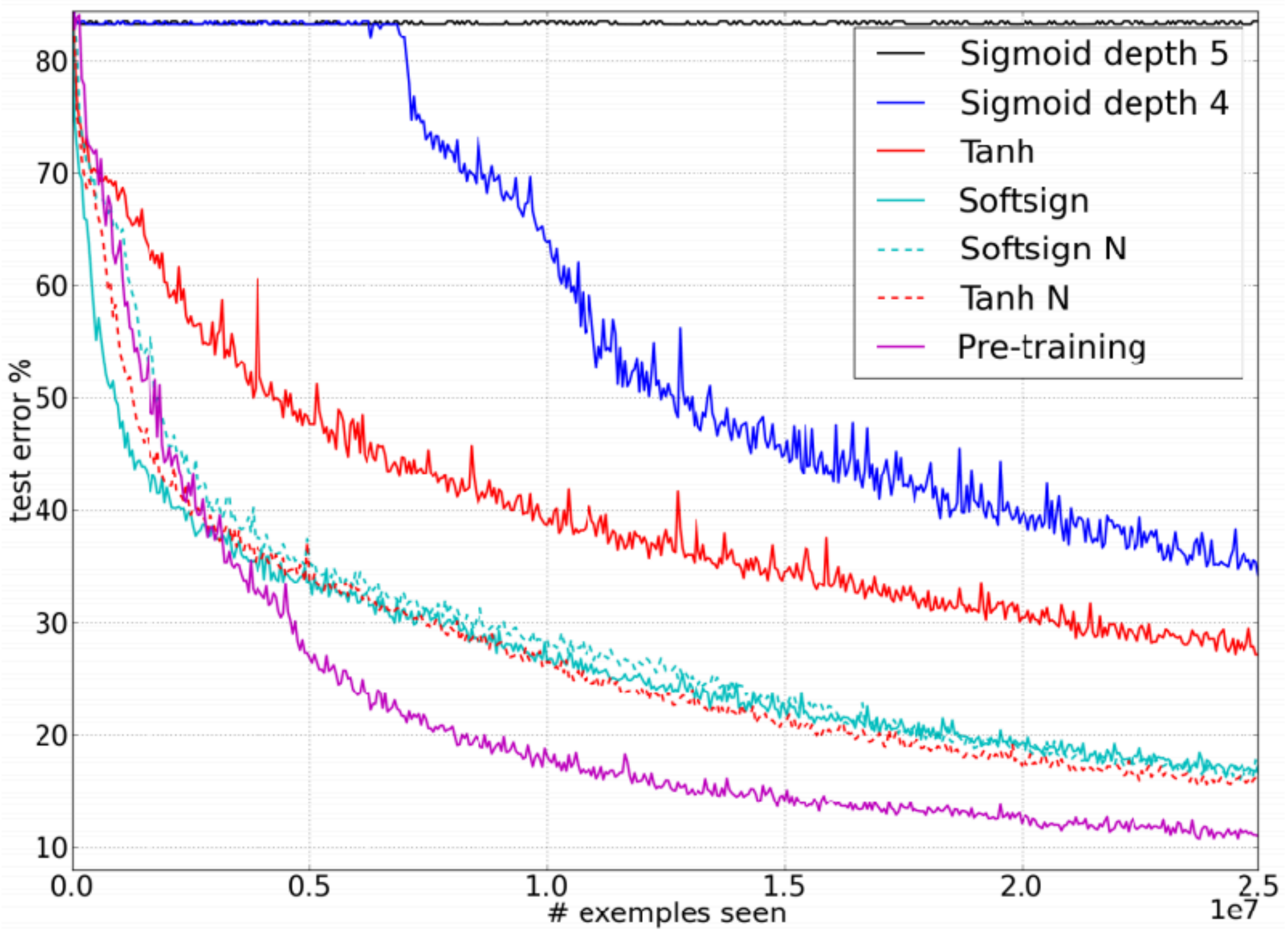


backpropagation

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

Solution 2 - normalized initialization(xavior initialization)

xavior initialization를 사용하지 않은 것과 비교하면
에러가 더 낮으며, 수렴도 더 빠르다는 것을 알 수 있다.



Tanh - hyperbolic tangent
Tanh N - hyperbolic tangent w/ xavior initialization

Understanding the difficulty of training deep feedforward neural networks
Glorot and Bengio, 2010. https://goo.gl/rFxJmU

TL; DR

- 같은 모델이라도 weight를 어떻게 초기화하냐에 따라 결과가 전혀 달라진다.
- weight는 기본적으로 random으로 초기화하지만, fan in과 fan out과 깊은 상관관계가 있다.

- sigmoid와 tanh에서는 xavior initialization을 사용한다.
- 하지만 공식을 유도해보면 알겠지만, ReLU와 같은 activate function에는 이를 동일하게 적용할 수 없다. **그렇다면 ReLU에는 어떤 공식을 적용해야 할까?** https://goo.gl/6fbfUx **이 링크를 참조**