딥러닝/클라우드

Chapter 10

Terms of DNN

Sejong Oh

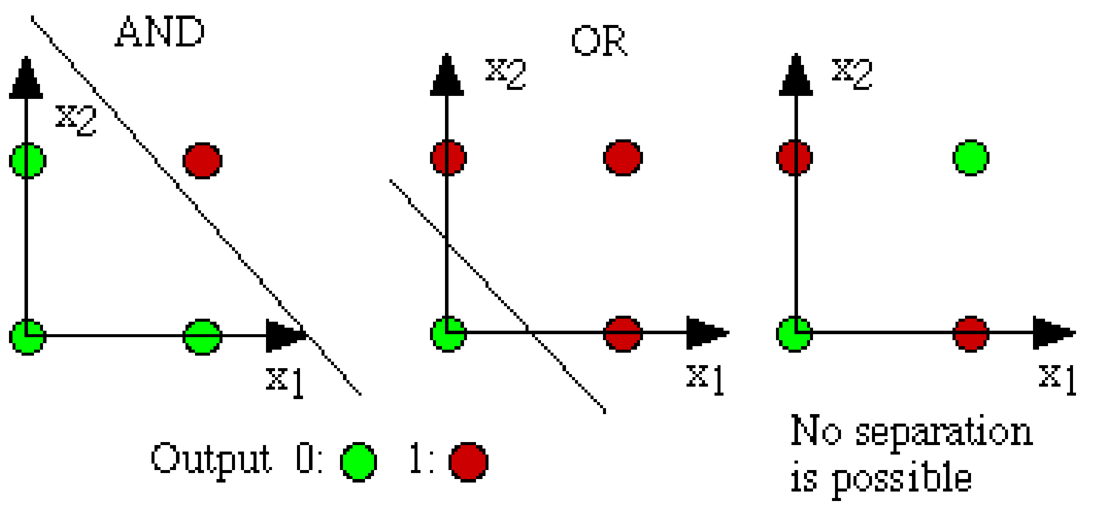
Bio Information technology Lab.

Contents

- Back propagation
- Momentum
- Drop out
- ReLU
- Initialize W

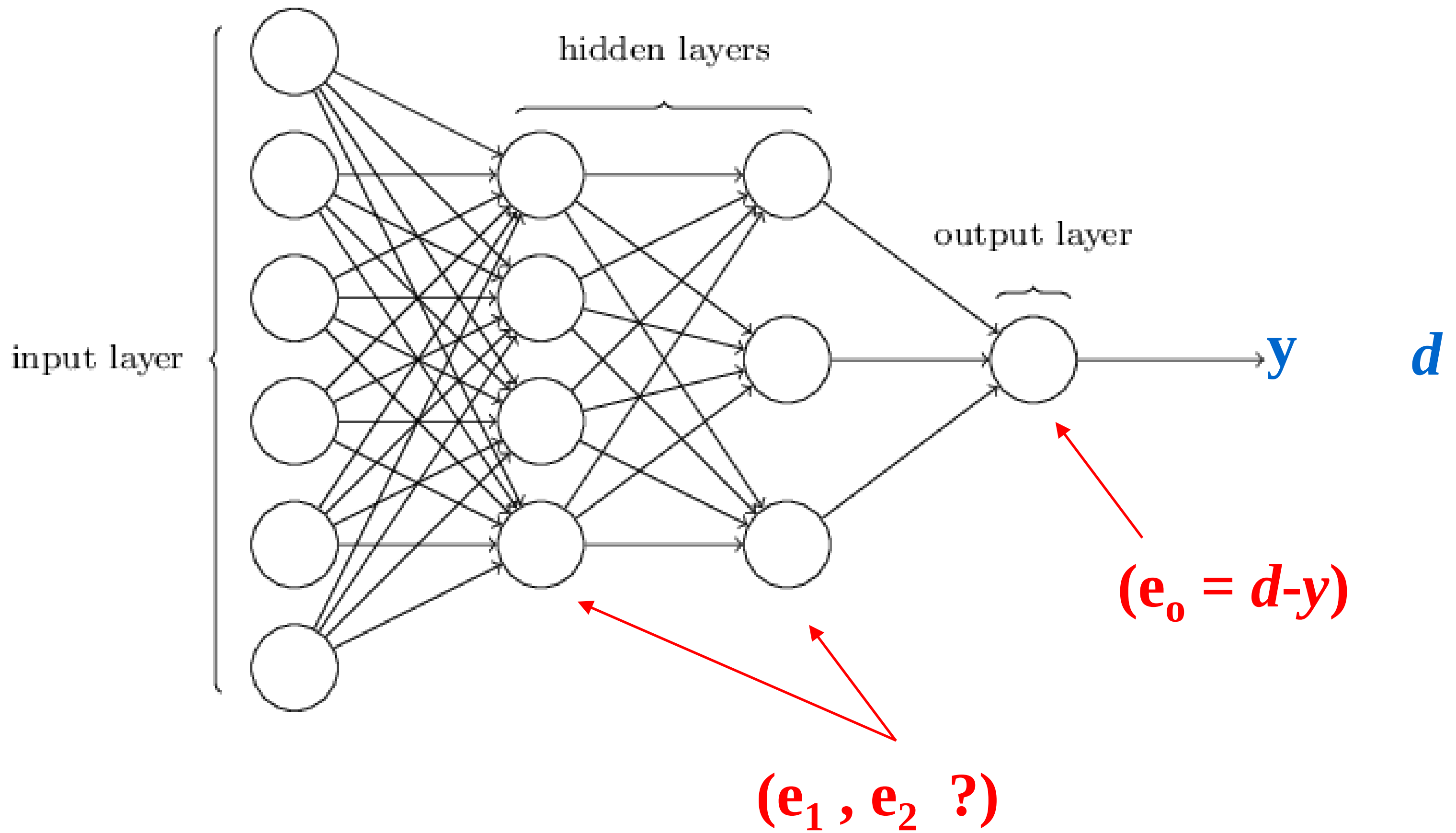
1. Back propagation

- Single layer perceptron can solve linear separation
  - Can not solve XOR problem



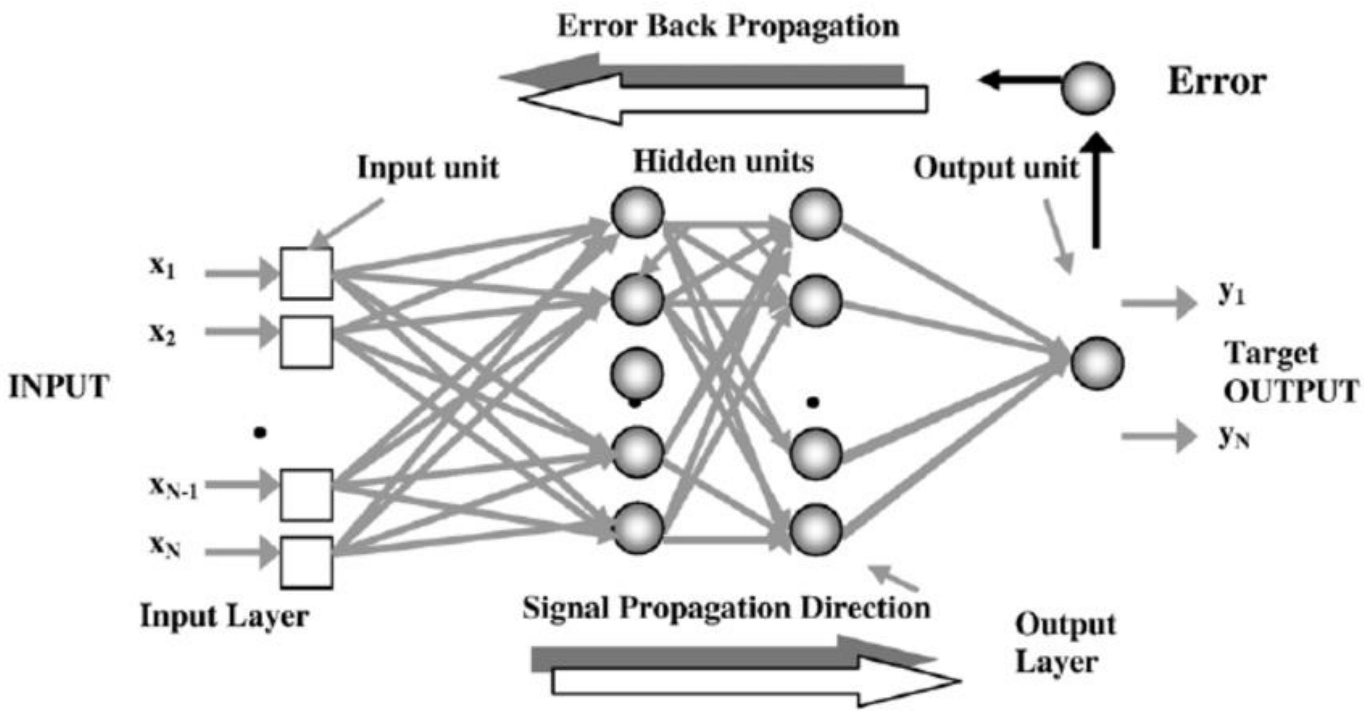
1. Back propagation

- Solution : multi-layer neural network
  - But can not find learning method
  - Delta rule requires error. We cannot define error of hidden layer



1. Back propagation

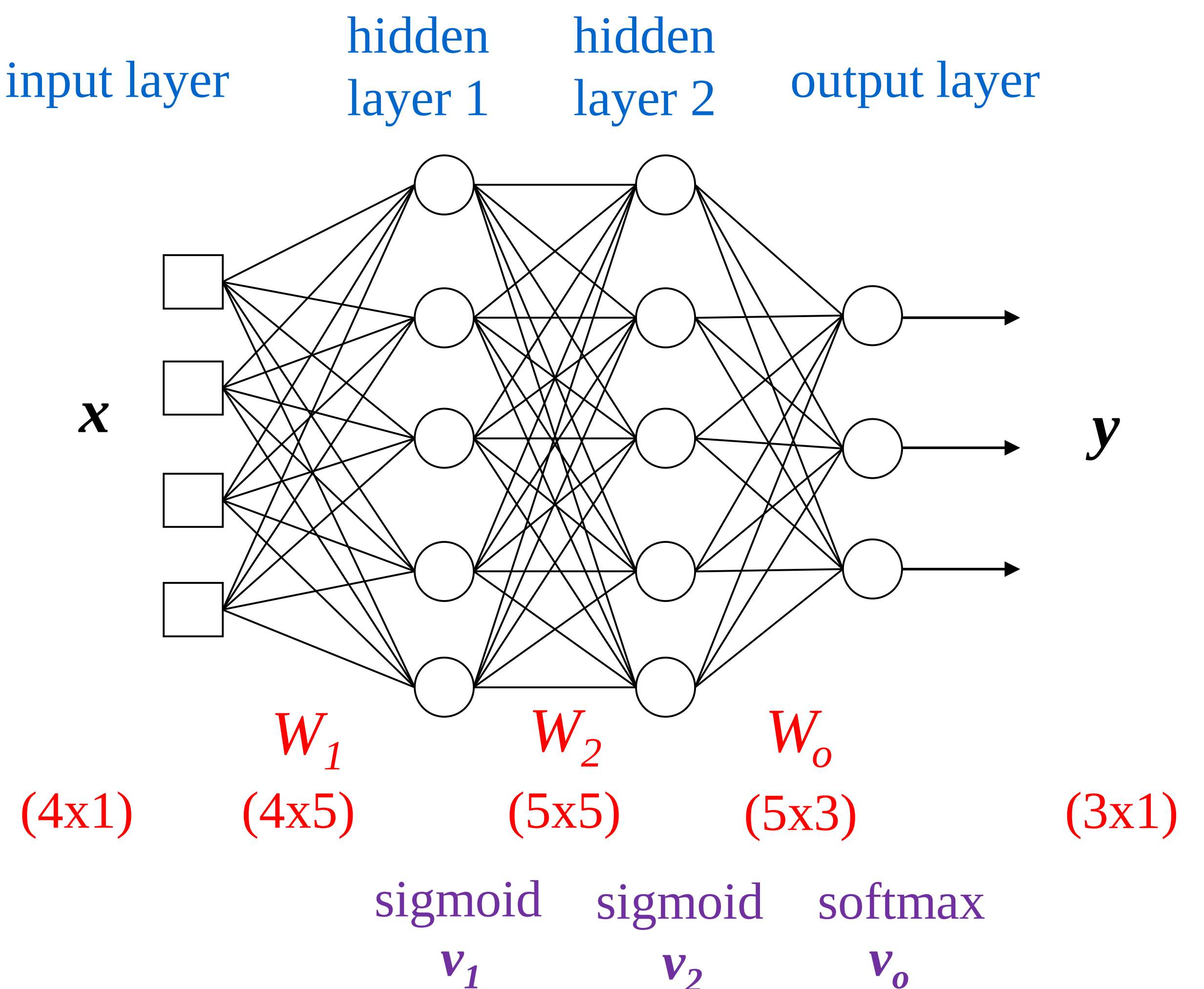
- Back propagation (역전파 알고리즘)
  - Define error and propagates it to backward



(feed forward)

1. Back propagation

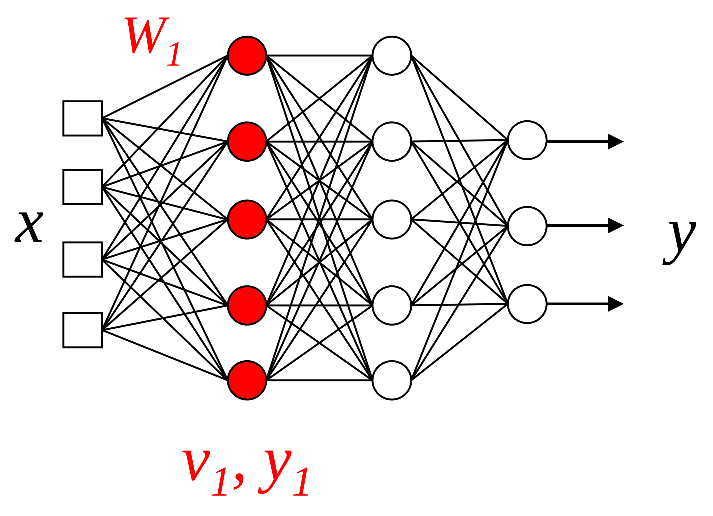
- Design multi-layer neural network to predict 'Species' of iris dataset
  - SGD update



1. Back propagation

- Feed forward

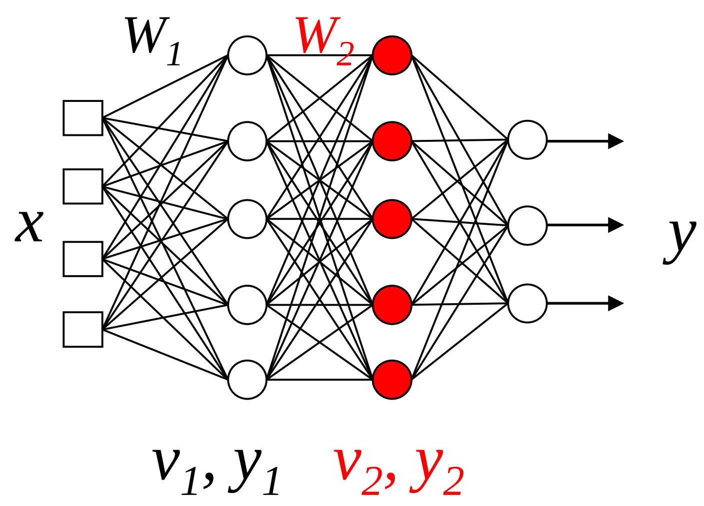
$$v_1 = xW_1^{\mathrm{T}}$$
$$y_1 = \mathrm{sigmoid}(v_1)$$



1. Back propagation

- Feed forward

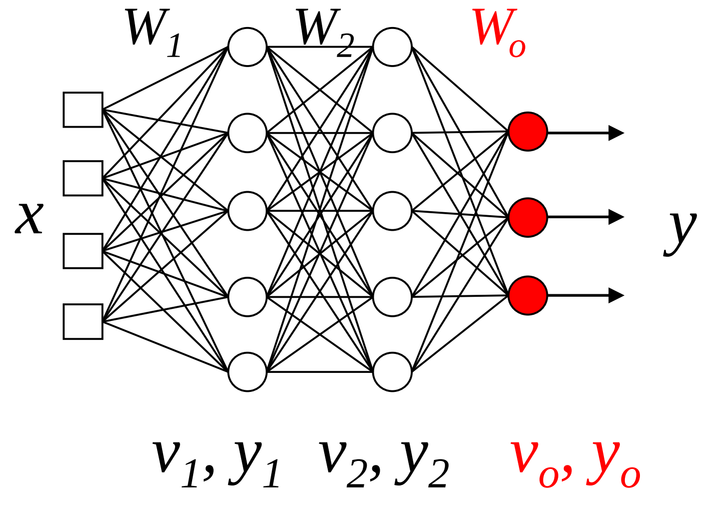
$$v_2 = y_1 W_2^{\mathrm{T}}$$
$$y_2 = \mathrm{sigmoid}(v_2)$$



1. Back propagation

- Feed forward

$$v_o = y_2 W_o^{\mathrm{T}}$$
$$y_o = \mathrm{softmax}(v_o)$$



1. Back propagation

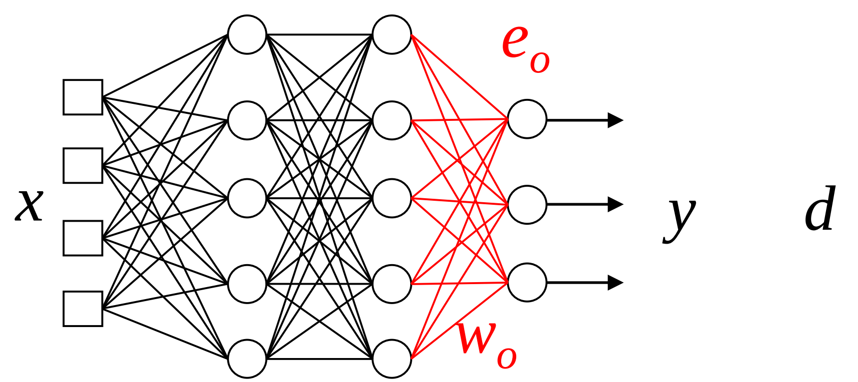
- Calculate e and δ, and update W_o

$$e_o = d - y_o$$

$$\delta_o = softmax'(v_o)\, e_o$$

$$\Delta w_o = \alpha\, \delta_o y_2$$

$$w_o \leftarrow w_o + \Delta w_o$$



1. Back propagation

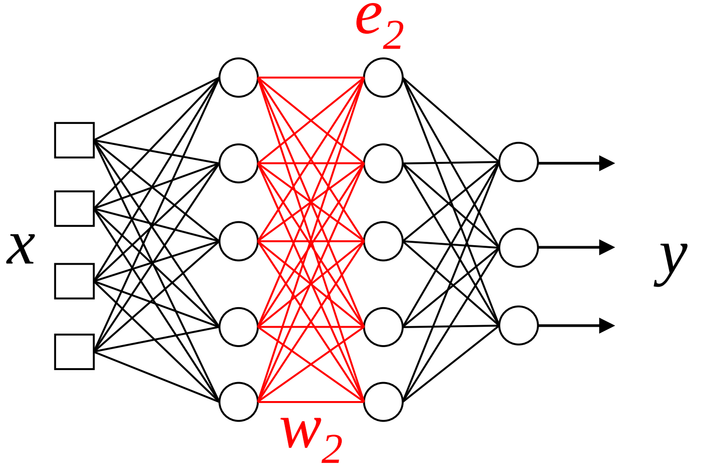
- Error back propagation

$$e_2 = W_o \times \delta_o$$

$$\delta_2 = sigmoid'(v_2)\, e_2$$

$$\Delta w_2 = \alpha\, \delta_2 y_1$$

$$w_2 \leftarrow w_2 + \Delta w_2$$



1. Back propagation

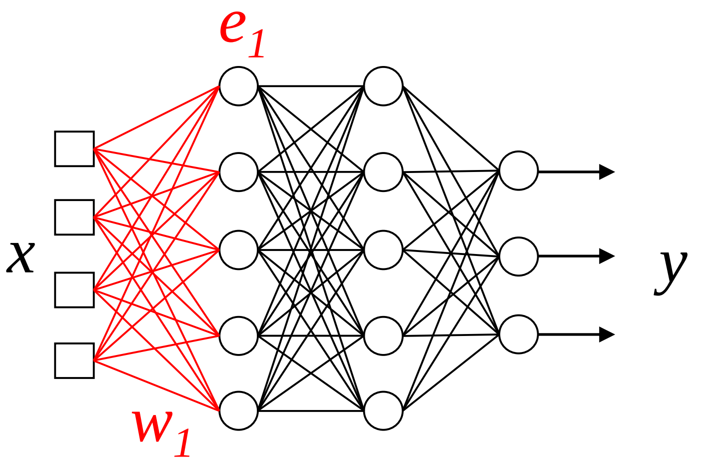
- Error back propagation

$$e_1 = W_2 \times \delta_2$$

$$\delta_1 = sigmoid'(v_1)\, e_1$$

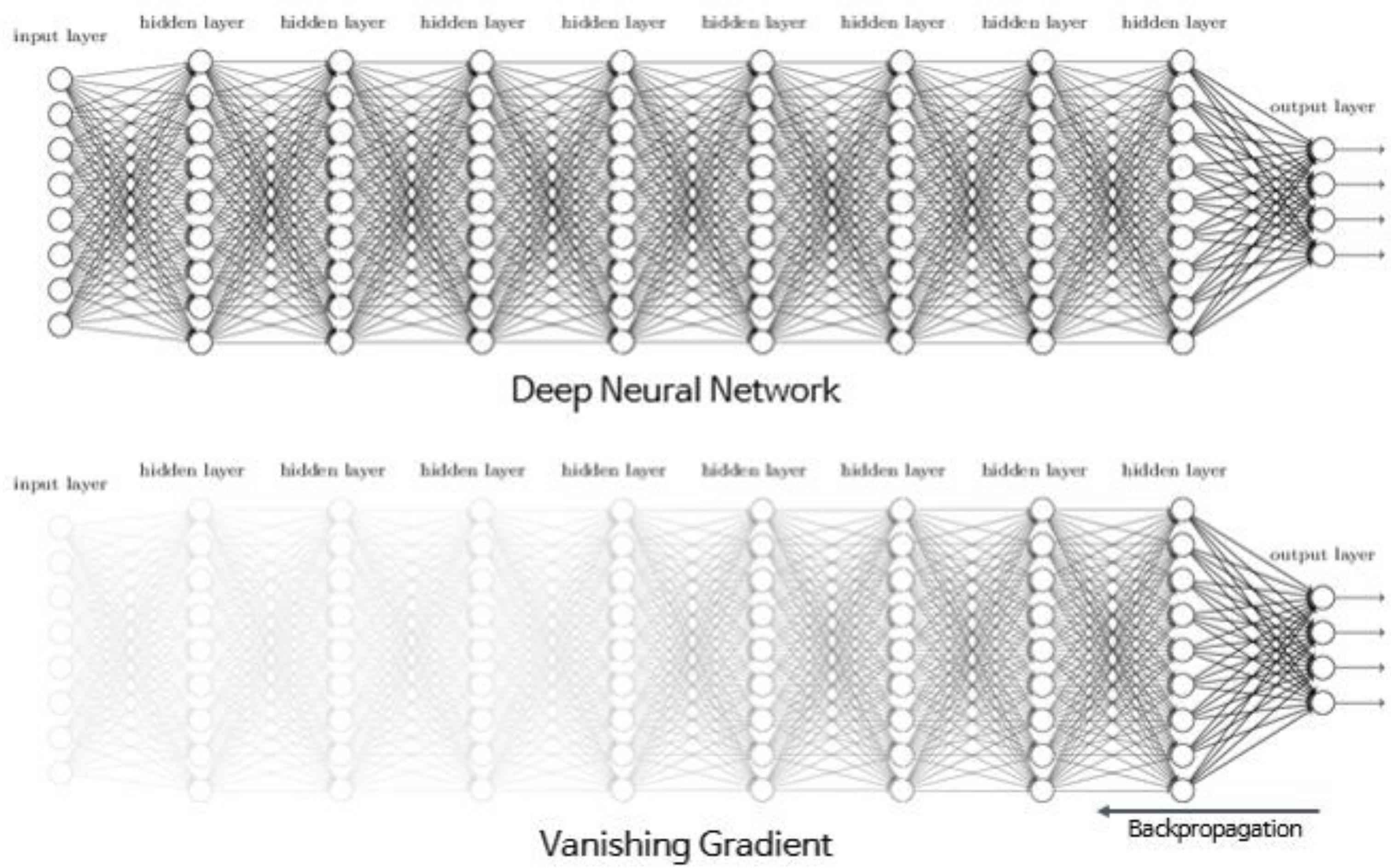
$$\Delta w_1 = \alpha\, \delta_1\, x$$

$$w_1 \leftarrow w_1 + \Delta w_1$$



1. Back propagation

- 기울기 소실 (vanishing gradient)문제



활성화 함수를 잘 골라야 한다

http://excelsior-cjh.tistory.com/177

2. ReLU

- One of activation function
- Sigmoid보다 계산 속도가 빠르기도 하지만 훨씬 더 빠르게 수렴
- 하지만 ReLU가 음수들을 모두 0으로 처리하기 때문에 한번 음수가 나오면 더이상 그 노드는 학습되지 않는다는 단점
- 이를 보완한 것이 Leaky ReLU

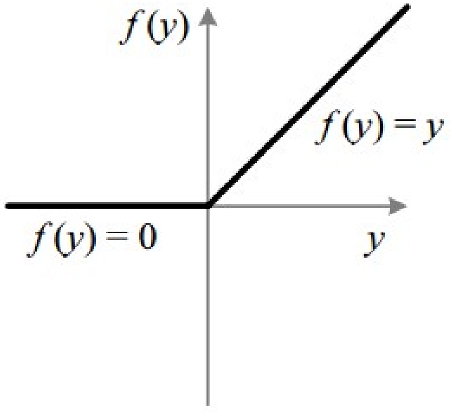
- Alleviate <u>gradient vanishing </u>in in deep neural network !

http://blog.naver.com/PostView.nhn?blogId=zzing0907&logNo=220693996517

2. ReLU

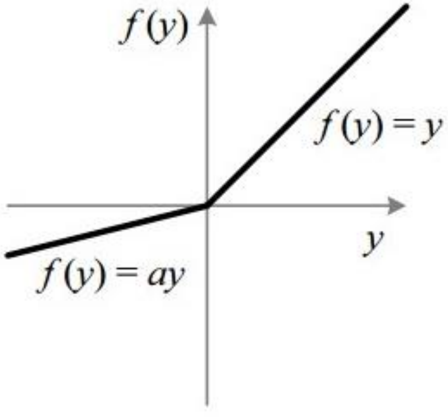
- ReLU

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



- Leaky ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

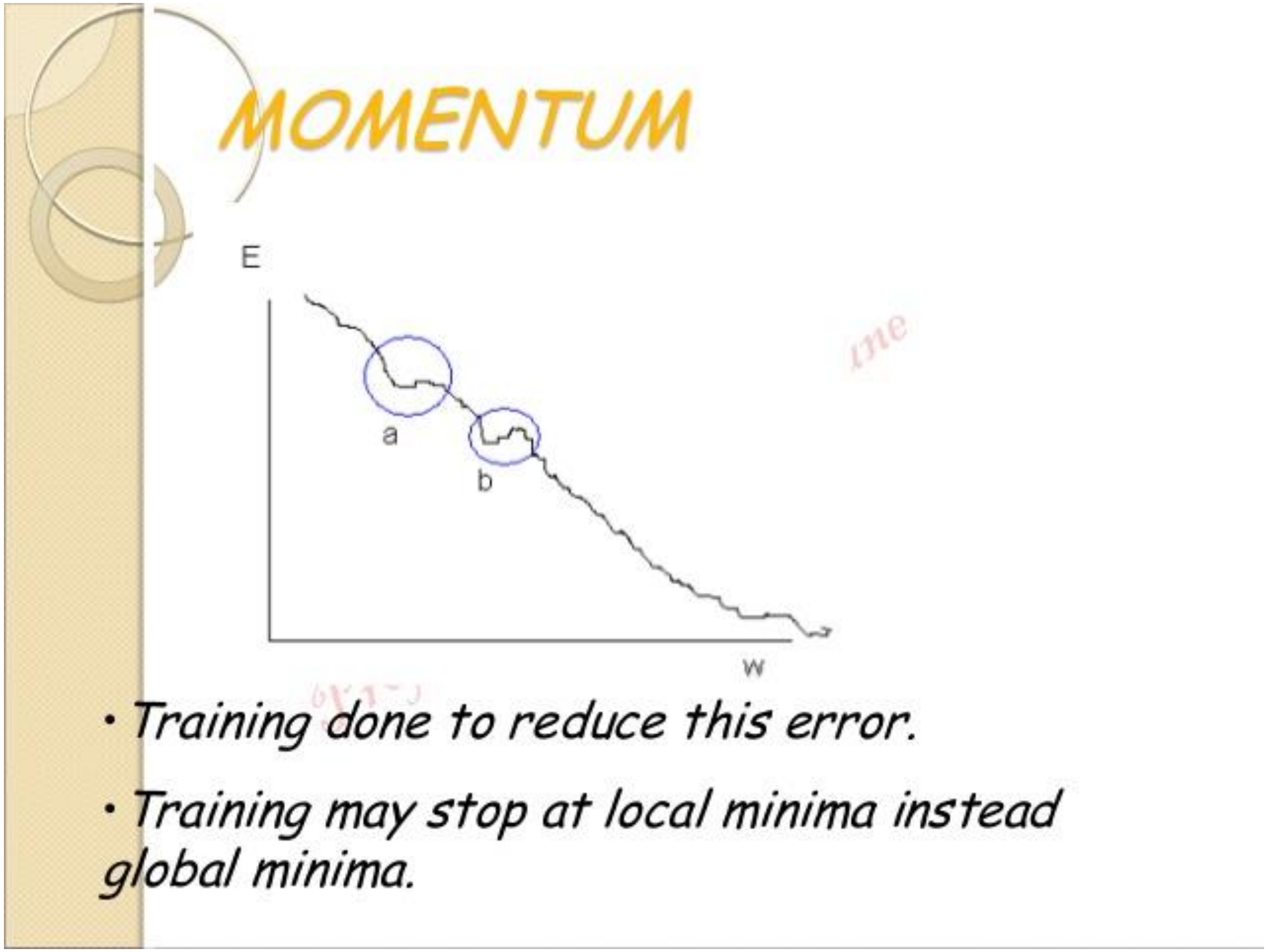


[Exercise 1]

- Implement ReLU and Leaky ReLU
- Test two functions using

$x =$ `-10, -5, -0.5, 0, 1, 5,10, 20, 100`

3. Momentum



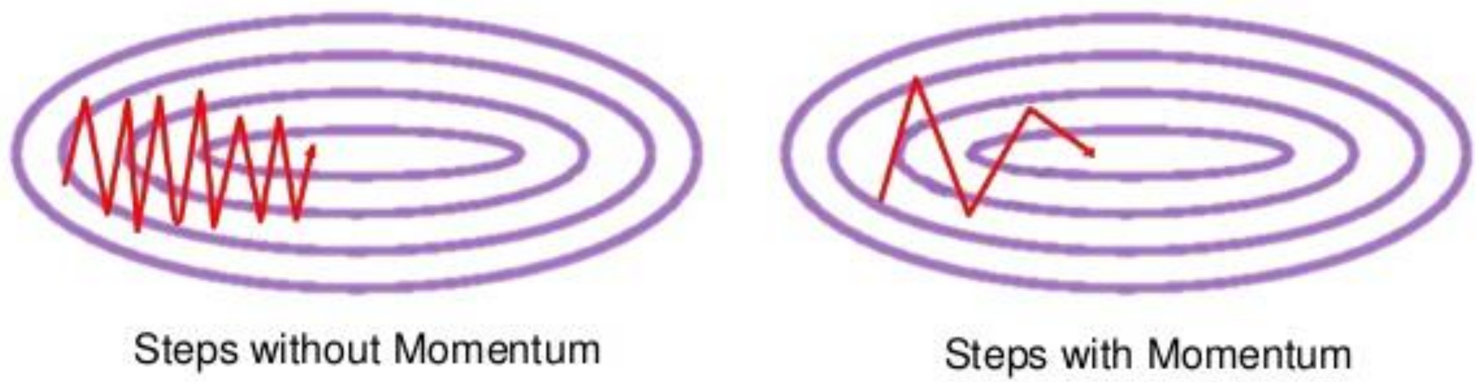
https://www.slideshare.net/MadhumitaTamhane/setting-artificial-neural-networks-parameters

3. Momentum

- The purpose of momentum is to speed up training but with a reduced risk of oscillating
- Control the degree of variance of weight matrix W

3. Momentum

- Momentum
  - Keep a running average of previous updates and add to each update



https://www.slideshare.net/embeddedvision/a-shallow-dive-into-training-deep-neural-networks-a-presentation-from-deepscale

3. Momentum

- General updating of W

$$\Delta w = \alpha \delta x$$

$$w \leftarrow w + \Delta w$$

3. Momentum

- updating of W with momentum

$$\Delta w = \alpha \delta x$$

$$m = \Delta w + \beta \bar{m}$$

$0 < \beta < 1$

$$w \leftarrow w + m$$

update weight

$$\bar{m} = m$$

update momentum

$\bar{m}$ **:** previous momentum

initial value of $\bar{m} = 0$

3. Momentum

- Note
  - 모멘텀은 가중치 갱신값을 새로 계산 할 때 델타 규칙외에 모멘텀을 추가로 더해서 가중치를 변경하는 방식
  - 가중치 갱신값이 바로 바뀌지 않고 어느정도 일정한 방향을 유지하면서 움직이게 됨
  - 관성을 가진 물체가 움직일 때 외부의 힘에 의해 쉽게 휘둘리지 않는 성질과 유사함
  - 현재 모멘텀은 과거 모멘텀 값이 계속 추가됨으로 해서 가중치 갱신값이 계속 커지게 됨. 이것이 학습속도가 향상되는 이유
  - β 값이 너무 크면 오히려 학습이 안되는 경우가 있다. (특히 SGD 방식)

4. Drop out*

- Neural networks are prone to overfitting

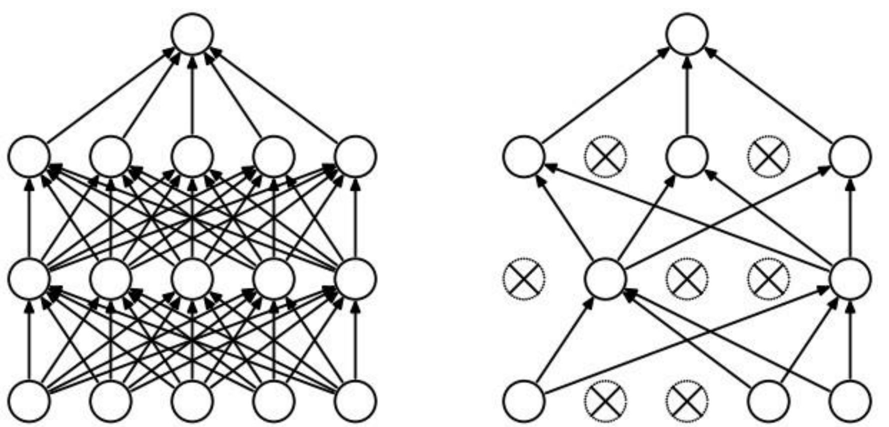
- 가중치 감쇠 (weight decay)
  - 신경망을 학습하는 과정에서 주어진 데이터에 딱 맞게 가중치를 조절하면 과적합이 발생할 수 있음
  - 가중치 W 를 갱신할 때 마다 0과 1 사이의 값을 가지는 가중치 ε 를 곱함으로써 학습을 방해하는 대신 과적합을 방지할 수 있다

$$W \leftarrow (W + \Delta W) \times (1 - \varepsilon)$$

4. Drop out*

- Drop out is a good solution for overfitting

- Dropout achieves better results than former used regularization methods (Weight Decay).
- Dropout is a bit slow to train (2-3 times slower than without dropout).
- If the amount of data is average-large – dropout excels. When data is big enough, dropout does not help much

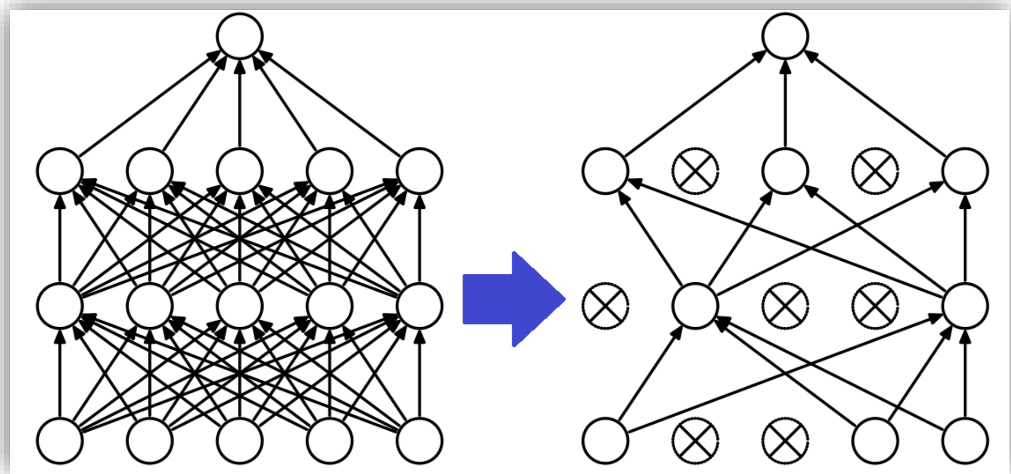


(a) Standard Neural Net (b) After applying dropout.

* http://web.eng.tau.ac.il/deep_learn/wp-content/uploads/2016/11/Deep-Learning-Seminar-Dropout.pptx

4. Drop out

- At training (each iteration):
  Each unit is retained with a probability p



- At test:
  The network is used as a whole.
  The weights are scaled-down by a factor of p.

4. Drop out

- The effect of the dropout rate p:

No dropout ($p = 1.0$)
- Training error is low
- Test error is high

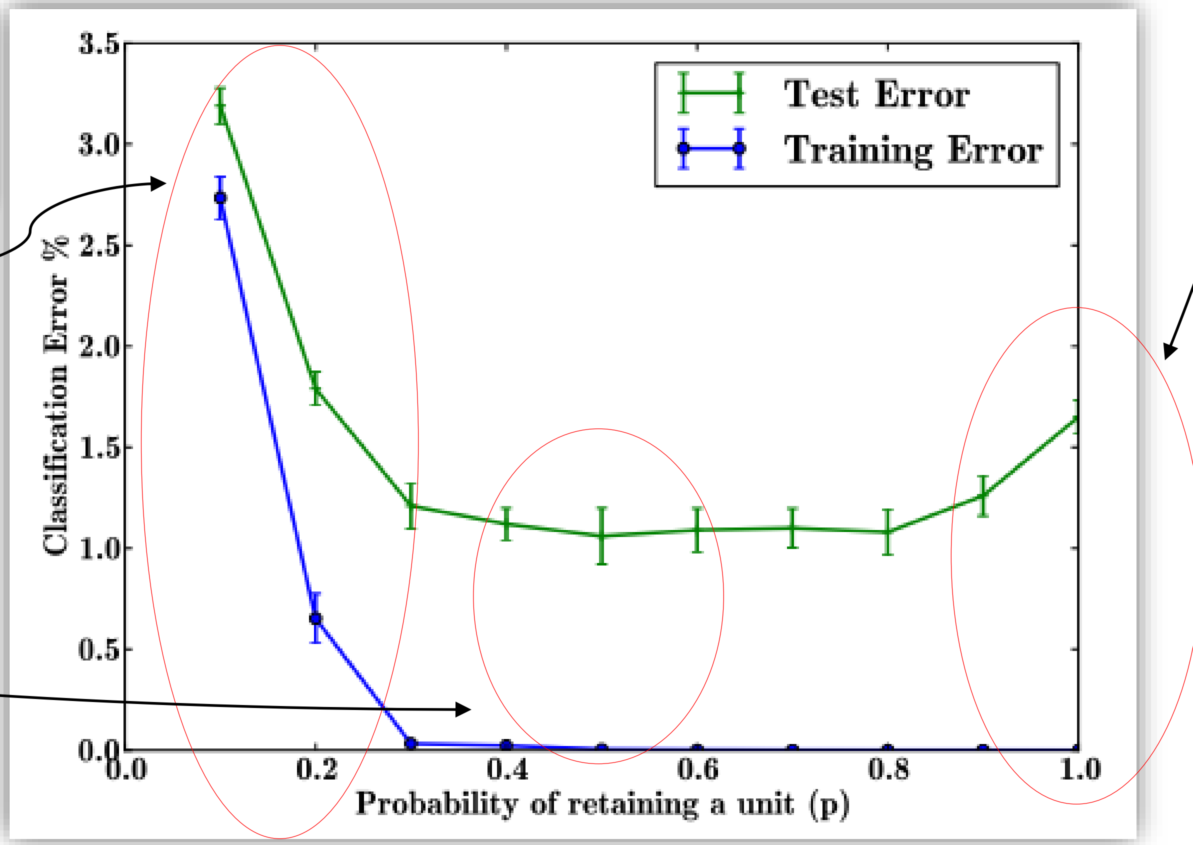
➔ Overfitting

High rate of dropout ($p < 0.3$)
- Training error is high

➔ Underfitting
- Very few units are turned on during training.

Best dropout rate ($p = 0.5$)
- Training error is low
- Test error is low

➔ Mission accomplished!



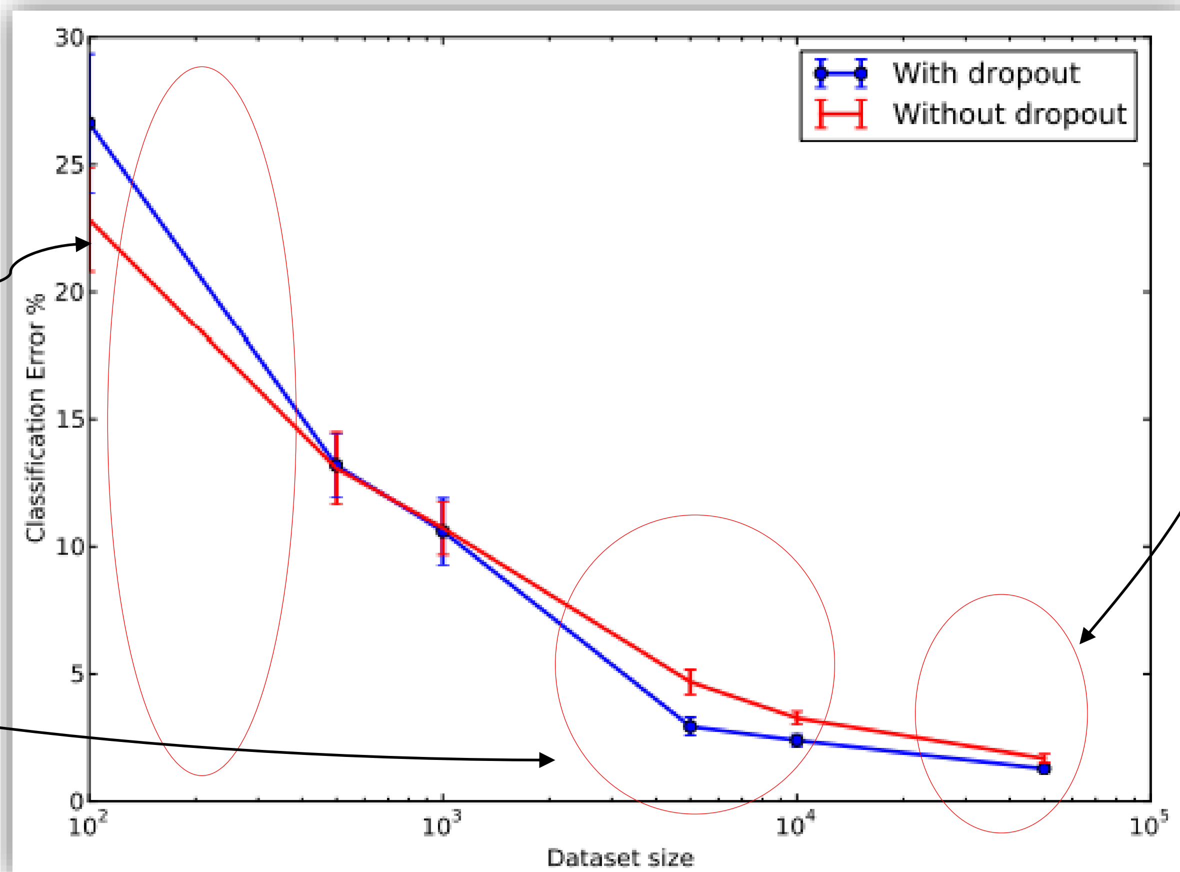
4. Drop out

- The effect of data set size

Huge data set
Dropout barely improves the error rate. The data set is big enough, so that overfitting is not an issue.

Extremely small data set
Dropout does not improve error rate, and even makes it worse.

Average to large data set
Dropout improves error rate.

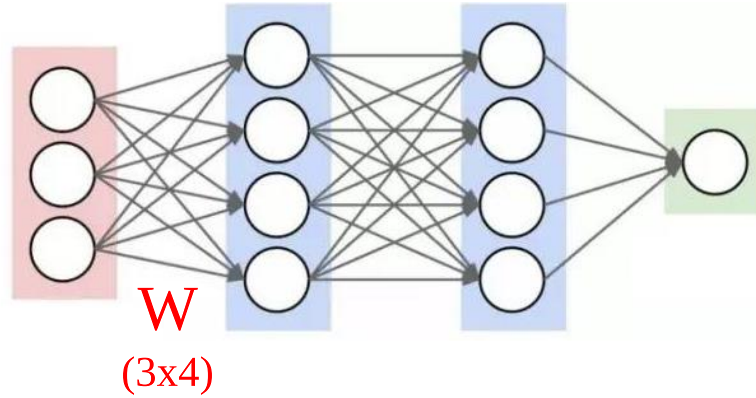


5. Initialize W

- Initializing W influences performance of neural network
- Random initializing is not good idea

- Xavier initialization
- He initialization

5. Initialize W



- Xavier initialize

```
import numpy as np
fan_in, fan_out = 3, 4
mn, sd = 0, 0.01   # mean, standard deviation
np.random.seed(123)
W_val = np.random.normal(mn, sd, fan_in*fan_out)/np.sqrt(fan_in)
W = W_val.reshape(fan_in,fan_out)
```

np.random.normal() : 정규분포를 만드는 함수

```
In [165]: W
Out[165]:
array([[-0.00626789,  0.00575818,  0.00163378, -0.0086966 ],
       [-0.00334055,  0.00953457, -0.01401044, -0.00247633],
       [ 0.00730889, -0.00500413, -0.00391955, -0.0005468 ]])
```

5. Initialize W

- He initialize

```
import numpy as np
fan_in, fan_out = 3, 4
mn, sd = 0, 0.01   # mean, standard deviation
np.random.seed(123)
W_val = np.random.normal(mn, sd, fan_in*fan_out)/np.sqrt(fan_in/2)
W = W_val.reshape(fan_in,fan_out)
```

```
In [167]: W
Out[167]:
array([[-0.00886414,  0.00814329,  0.00231051, -0.01229884],
       [-0.00472425,  0.01348392, -0.01981375, -0.00350206],
       [ 0.01033633, -0.00707691, -0.00554308, -0.0007733 ]])
```

5. Initialize W

- Activation function : ReLU
  ➡ Use He initialization

- Activation function : sigmoid, tanh
  ➡ Use Xavier initialization

Keywords

- Back propagation
- Vanishing gradient problem
- ReLU
- Momentum
- Drop out
- He, Xavier initialization