

# **머신러닝 개요**

## **Lecture 7: Optimizing Neural Net**

**College of Information and Electronic Engineering**

**Kyung Hee University**

**Prof. Wonha Kim**

**([wonha@khu.ac.kr](mailto:wonha@khu.ac.kr))**

# Contents

- **Data Processing** : Normalization, One hot encoding
- **Object functions** : MSE, CEE
- **Activation functions** : softmax, LeRu, Tanh, Sigmoid
- **Optimizer** : Momentum method, Adaptive Learning Rate method
- **Weight Initialization**
- **Preventing Overfitting** :
- **Batch Normalization**

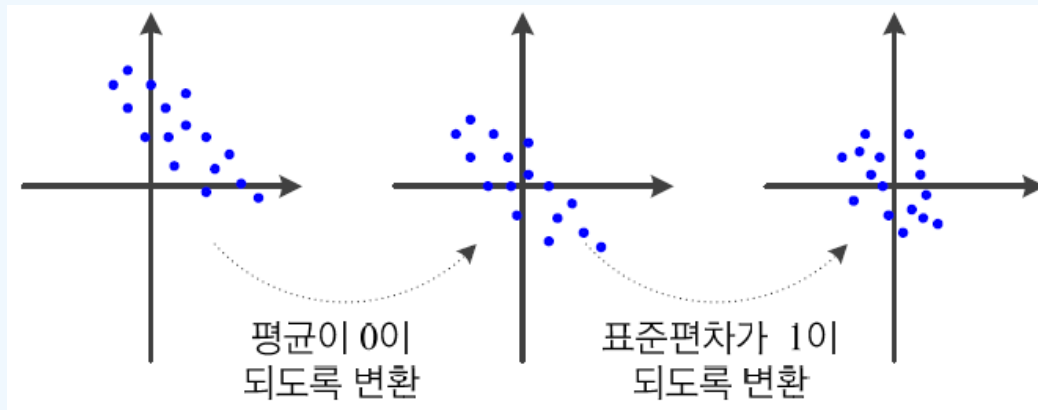


# 1. Data preprocessing

- Normalization

- Normalization handles scale and unbalanced weights.

$$x_i^{new} = \frac{x_i^{old} - \mu_i}{\sigma_i}$$



- One-hot encoding

- Certain features have names instead of values, that is called as “nominal value” .
- Example: wind direction = { E,W,S,N } => 2 bits.

x=(wind velocity, direction)->[100,1,0]

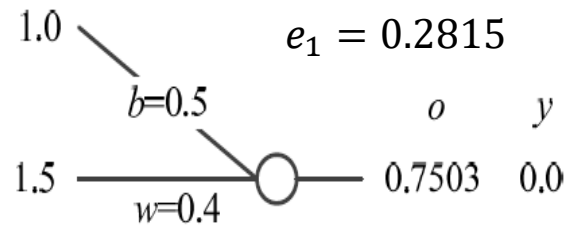
Two bits for direction

## 2. Object function (1/2): Mean Squared Error (MSE)

- Training process updates the weights (parameters) to reduce learning errors.  
From the update equation, the gradient is the update value, or the penalty.
- For larger learning error, weights should be assigned by large penalty (gradient).

$$w \leftarrow w - \eta \frac{\partial J}{\partial W} \leftarrow \text{Penalty}$$

- **MSE :  $J = \|Y - O\|_2^2$**



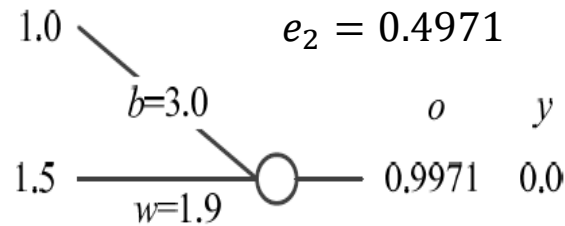
$$\frac{\partial e}{\partial w} = 0.2109 \quad \frac{\partial e}{\partial b} = 0.1406$$

$$e = \frac{1}{2}(y - o)^2 = \frac{1}{2}(y - \sigma(wx + b))^2$$

$$\frac{\partial e}{\partial w} = -(y - o)x\sigma'(wx + b)$$

$$\frac{\partial e}{\partial b} = -(y - o)\sigma'(wx + b)$$

$$\frac{\partial e}{\partial w} = 0.75 * 1.5 * \sigma'(0.4 * 1.5 + 0.5) = 1.12 + 0.21 = 0.21$$



$$\frac{\partial e}{\partial w} = 0.0043 \quad \frac{\partial e}{\partial b} = 0.0029$$

**Error :**  $e_1 = 0.2815 < e_2 = 0.4971$

**Gradient :**  $G1 = 0.21 + 0.14 > G2 = 0.004 + 0.002$

In MSE, larger error has smaller penalty.

MSE is simply and seems to be reasonable for error measure. But, it may be inefficient in learning process.



## 2. Object function (2/2): Cross Entropy Error (CEE)

- Cross Entropy Error (CEE)** :  $e = -(y \log_2 o + (1 - y) \log_2(1 - o))$  where  $o = \sigma(z)$  and  $z = wx + b$

### Example

If  $y=1, o= 0.98$  (well estimated)

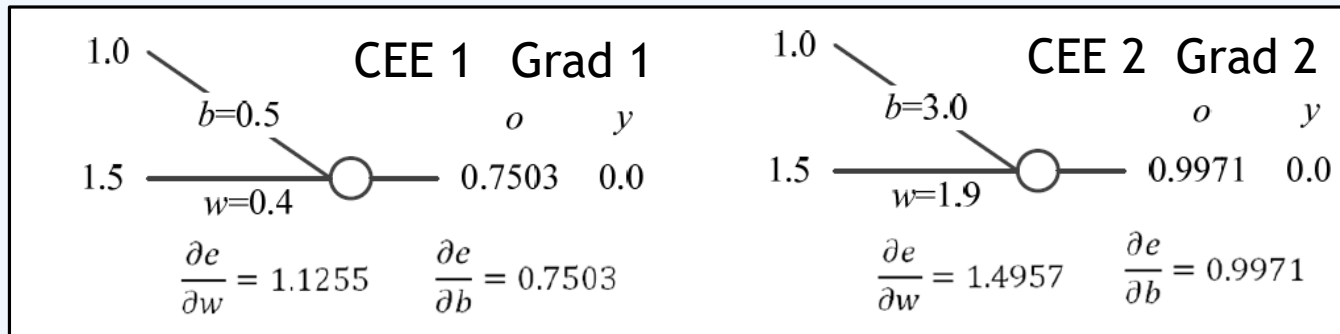
$$e = -(1 \log_2 0.98 + (1 - 1) \log_2(1 - 0.98)) = 0.0291$$



If  $y=1, o= 0.0001$  (Poorly estimated)

$$e = -(1 \log_2 0.0001 + (1 - 1) \log_2(1 - 0.00001)) = 13.28$$

- Gradient** :  $\frac{\partial e}{\partial w} = -\left(\frac{y}{o} - \frac{1-y}{1-o}\right) \frac{\partial o}{\partial w} = -\left(\frac{y}{o} - \frac{1-y}{1-o}\right) x \sigma'(z) = -x \left(\frac{y}{o} - \frac{1-y}{1-o}\right) o(1-o) = x(o-y) \Rightarrow \frac{\partial e}{\partial w} = x(o-y) \quad \frac{\partial e}{\partial b} = (o-y)$

- Simple case** :



- Error : CEE 1 < CEE2
  - Gradient : Grad 1 < Grad 2
-   
**In CEE, larger error has larger penalty**  
  
**CEE is more efficient than MSE.**

- Generally, # of nodes is  $c$ , that is,  $\mathbf{o} = (o_1, o_2, \dots, o_c)^T$

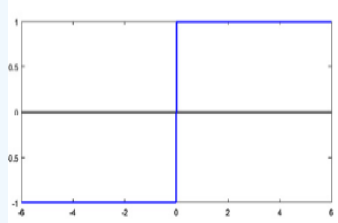
$$e = - \sum_{i=1,c} (y_i \log_2 o_i + (1 - y_i) \log_2(1 - o_i))$$



# 3. Activation functions (1/2) : ReLU

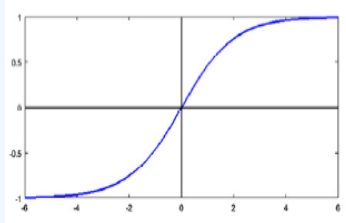
- Evolution of activation functions

– 1950s



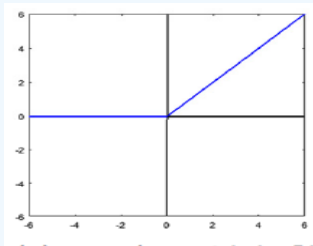
- Step

– 1980s



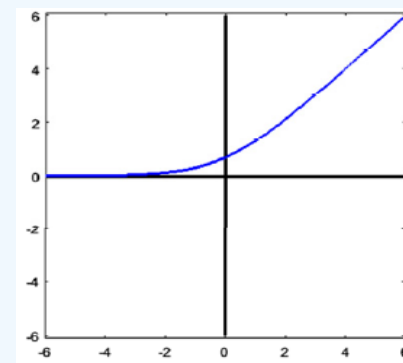
- **TANH**  
At saturation ranges ( $|z| > 10$ ), the derivate is almost zero, so, causing very slow convergence.

– 2000~current

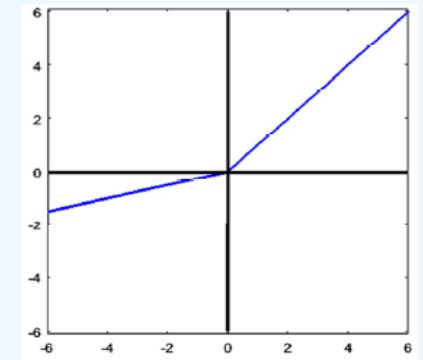


- **ReLU**

- Rectified Linear Unit (ReLU) :  $y = \text{ReLU}(z) = \max(0, z)$ 
  - No saturation.
  - Impossible differentiation at zero, but NN needs values closer to zero instead of zero itself. ReLU is even better than softplus that is differentiable at all values.
  - Modified ReLU



softplus



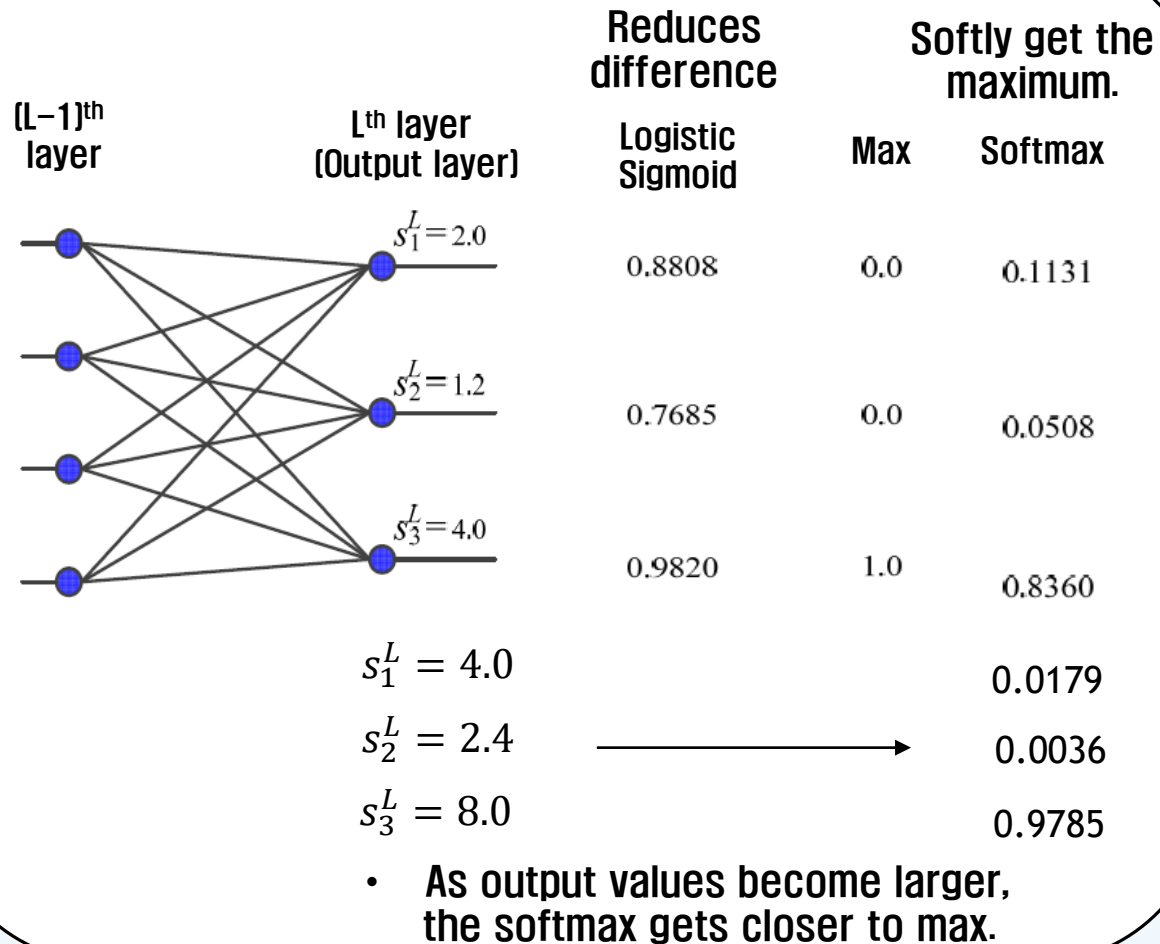
$$\text{leakyReLU}(z) = \begin{cases} z, & z \geq 0 \\ \alpha z, & z < 0 \end{cases}$$

(usually  $\alpha=0.01$ )



### 3. Activation function (2/2): Softmax

- **Softmax** :  $o_j = \frac{e^{s_j}}{\sum_{i=1,c} e^{s_i}}$ 
  - Softly get the maximum value.
  - Sum of output node value is 1. So. can be uses as probability.



- Often used as activation. Combined with other object function.
  - Softmax + ECC
  - Softmax + MSE
  - Softmax + Log Likelihood (well match and so frequently used)
- **Log Likelihood** :  $e = -\log_2 o_y$ 
  - Unlike MSE or CEE using all output node values, use only one node value.
  - Example : Softmax+Log Likelihood

$$e = -\log_2 0.1131 = 3.1443$$

$$e = -\log_2 0.0508 = 4.2990$$

$$e = -\log_2 0.8360 = 0.2584$$

Smallest error



## 2. Techniques for performance improvement

Table 11-2. Default DNN configuration

Initialization	He Initialization
Activation function	ELU
Normalization	Batch Normalization
Regularization	Dropout
Optimizer	Adam
Learning rate schedule	None

Of course, you should try to reuse parts of a pretrained neural network if you can find one that solves a similar problem.

This default configuration may need to be tweaked:

- If you can't find a good learning rate (convergence was too slow, so you increased the training rate, and now convergence is fast but the network's accuracy is sub-optimal), then you can try adding a learning schedule such as exponential decay.
- If your training set is a bit too small, you can implement data augmentation.
- If you need a sparse model, you can add some  $\ell_1$  regularization to the mix (and optionally zero out the tiny weights after training). If you need an even sparser model, you can try using FTRL instead of Adam optimization, along with  $\ell_1$  regularization.
- If you need a lightning-fast model at runtime, you may want to drop Batch Normalization, and possibly replace the ELU activation function with the leaky ReLU. Having a sparse model will also help.





## 4. Optimizer (1/3) : Momentum method

- Issue of Stochastic Gradient Descent(SGD) :  
Gradient inherently cause noise or overshooting.

- Momentum method :

- Momentum method adds smoothing to gradient to reduce the gradient noise effects (i.e., overshooting) and so increase the learning speed.

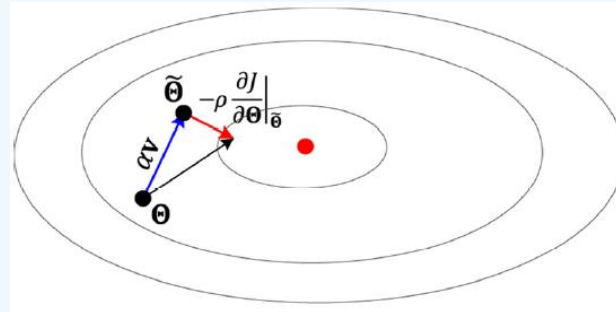
$$v \leftarrow \alpha v - \eta \frac{\partial J}{\partial W} \quad w \leftarrow w + v \quad \text{점프가 크면 클수록 momentum 감소}$$

- Velocity  $v$  is the accumulation of previous gradients. Initially,  $v=0$ .
- As  $\alpha$  approaches 1, previous gradients are more counted and so the trajectory of  $w$  is more smooth.
- Usually,  $\alpha$  is set by 0.5, 0.9 or 0.99. Or starting at 0.5, as learning proceeds, we let it gradually reach to 0.99.

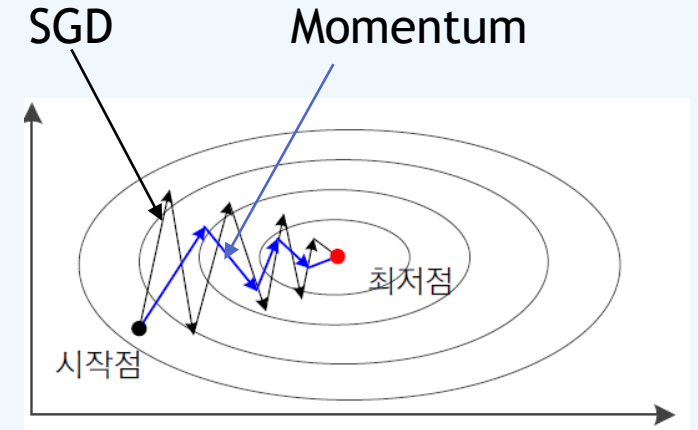
- Nesterov Momentum

- From  $v$ , predict the next location of the current  $w$ . Then, use the gradient at the predicted location.

$$\begin{aligned} \hat{w} &\leftarrow w + \alpha v \\ v &\leftarrow \alpha v - \eta \frac{\partial J}{\partial W} \Big|_{\hat{w}} \\ w &\leftarrow w + v \end{aligned}$$

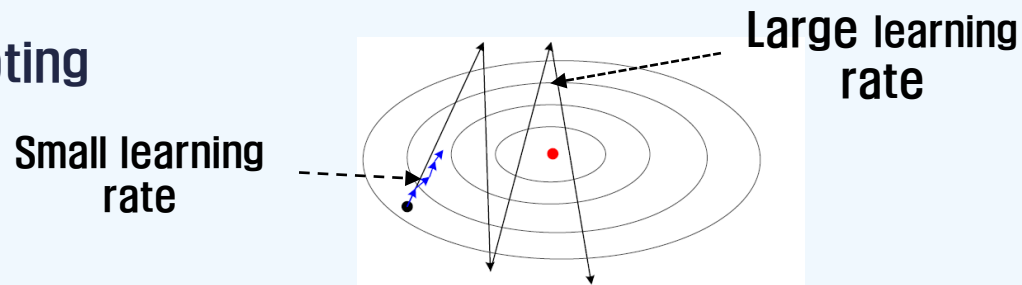


알파, 뮤 = 고정



## 4. Optimizer (2/3) : Adaptive Learning Rate

- Issue of learning rate  $\eta$  : Large  $\eta$  causes overshooting and small  $\eta$  causes slow convergence.



- Adaptive Gradient (AdaGrad) method
  - Gradient Adaptive learning rate decay.

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial J}{\partial \mathbf{W}} \circ \frac{\partial J}{\partial \mathbf{W}} \quad \Rightarrow \quad h_i \leftarrow h_i + \left| \frac{\partial J}{\partial w_i} \right|^2 \quad \mathbf{h} = [h_1 \dots h_K] : \text{vector accumulating the previous gradients.}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{\epsilon + \sqrt{\mathbf{h}}} \frac{\partial J}{\partial \mathbf{W}} \quad \Rightarrow \quad w_i \leftarrow w_i - \eta \frac{1}{\epsilon + \sqrt{h_i}} \frac{\partial J}{\partial w_i} : \text{For small } h_i, w_i \text{ moves much. For large } h_i, w_i \text{ moves less.}$$

Small number preventing overflow

- Adaptive Moment Estimation (Adam) method
  - Momentum + AdaGrad

$$\begin{aligned} \mathbf{m} &\leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \frac{\partial f}{\partial \mathbf{W}} & \hat{\mathbf{m}} &\leftarrow \mathbf{m} / (1 - \beta_1^2) \\ \mathbf{v} &\leftarrow \beta_2 \mathbf{v} + (1 - \beta_2) \frac{\partial f}{\partial \mathbf{W}} & \hat{\mathbf{v}} &\leftarrow \mathbf{v} / (1 - \beta_2^2) \end{aligned} \quad \Rightarrow \quad \mathbf{w} \leftarrow \mathbf{w} - \eta \hat{\mathbf{m}} / (\hat{\mathbf{v}} + \epsilon)$$

# 4 Optimizer (3/3) :Summary

- Stochastic Gradient Descent(SGD) method

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial f}{\partial \mathbf{W}}$$

- Momentum method

- Accelerate the learning velocity.

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \eta \frac{\partial f}{\partial \mathbf{W}} \quad \mathbf{w} \leftarrow \mathbf{w} + \mathbf{v}$$

- Adaptive Gradient (AdaGrad) method

- Gradient Adaptive learning rate decay.

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial f}{\partial \mathbf{W}} \circ \frac{\partial f}{\partial \mathbf{W}} \quad \mathbf{w} \leftarrow \mathbf{w} - \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial f}{\partial \mathbf{W}}$$

- Adaptive Moment Estimation (Adam) method

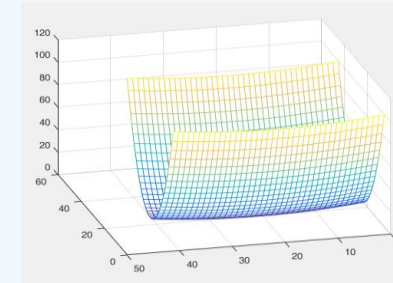
- Momentum + AdaGrad

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1 - \beta_1) \frac{\partial f}{\partial \mathbf{W}} \quad \hat{\mathbf{m}} \leftarrow \mathbf{m} / (1 - \beta_1^2) \quad \mathbf{w} \leftarrow \mathbf{w} - \eta \hat{\mathbf{m}} / (\hat{\mathbf{v}} + \epsilon)$$

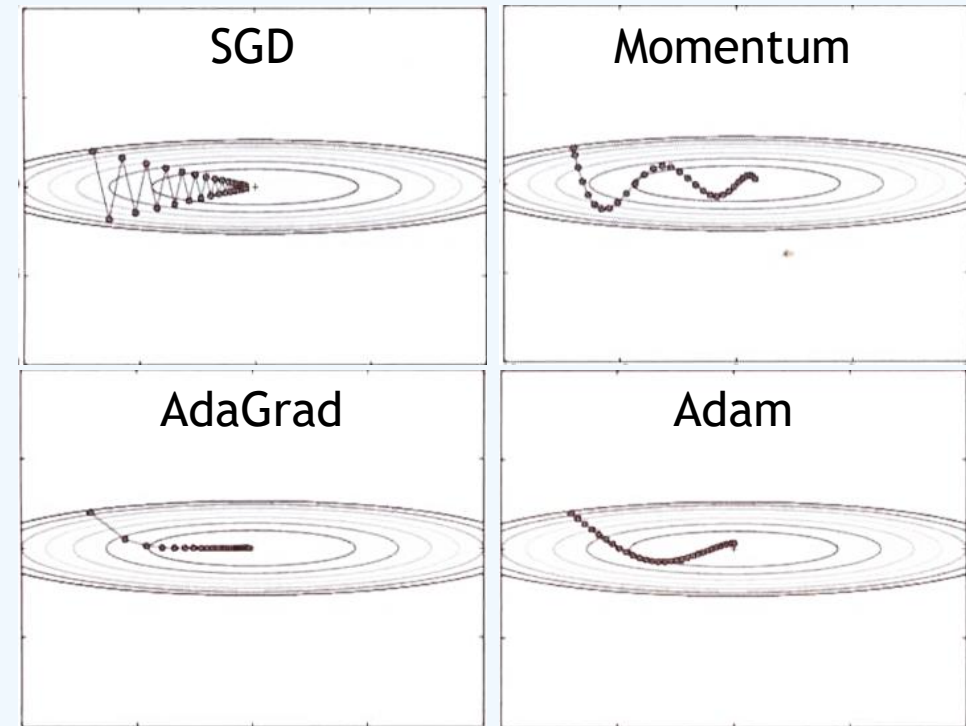
$$\mathbf{v} \leftarrow \beta_2 \mathbf{v} + (1 - \beta_2) \frac{\partial f}{\partial \mathbf{W}} \quad \hat{\mathbf{v}} \leftarrow \mathbf{v} / (1 - \beta_2^2)$$

주로 아담을 많이 씀  
그냥 직관적으로 이해해보자

- Comparison of optimizers



$$f(x, y) = \frac{1}{20} x^2 + y^2$$

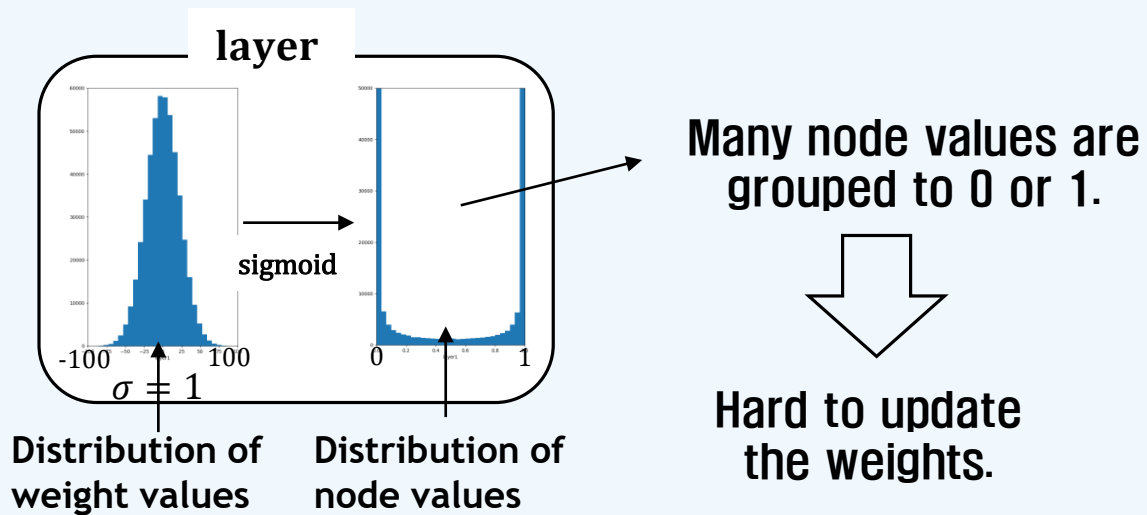


# 5. Weights Initialization (1/2) : Effects of weight distribution

- Initial weight values should be very small for preventing overfitting, but can not be zeros.
- Initial weight values should be random or not concentrated on certain range so that all weights are equally updated in backward propagation.

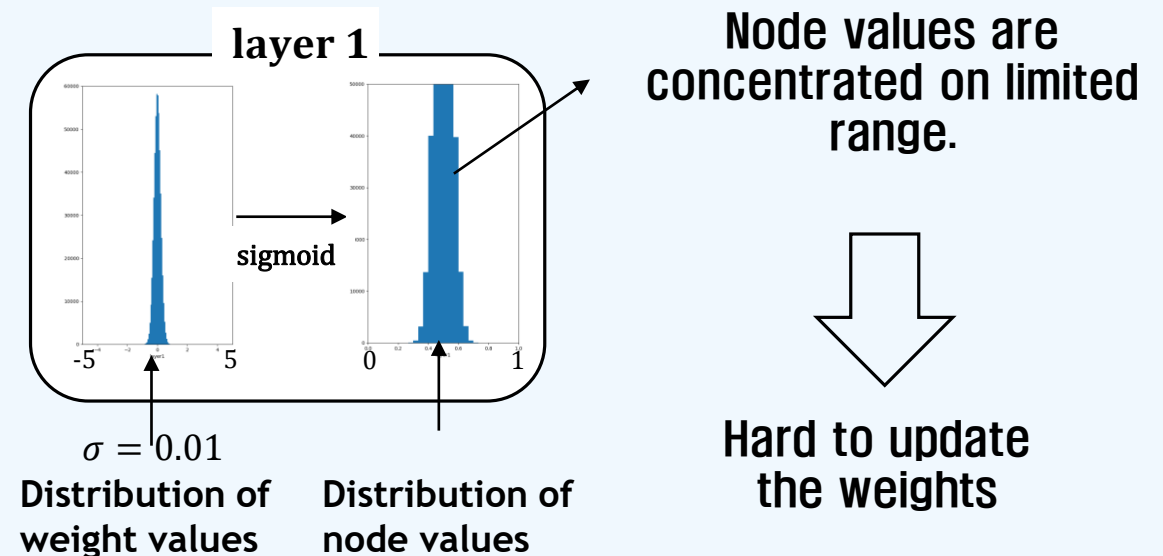
## Gradient vanishing : Too large of range

- If gradients of node values are very small, the network is not efficiently trained.
- For example, as outputs of the sigmoid approach to 0 or 1, the gradient becomes more closer to 0 and so the weights are rarely updated.



## Excessive Point: Too small of range

- If the weight values are grouped on certain range, the outputs of nodes are also clustered.

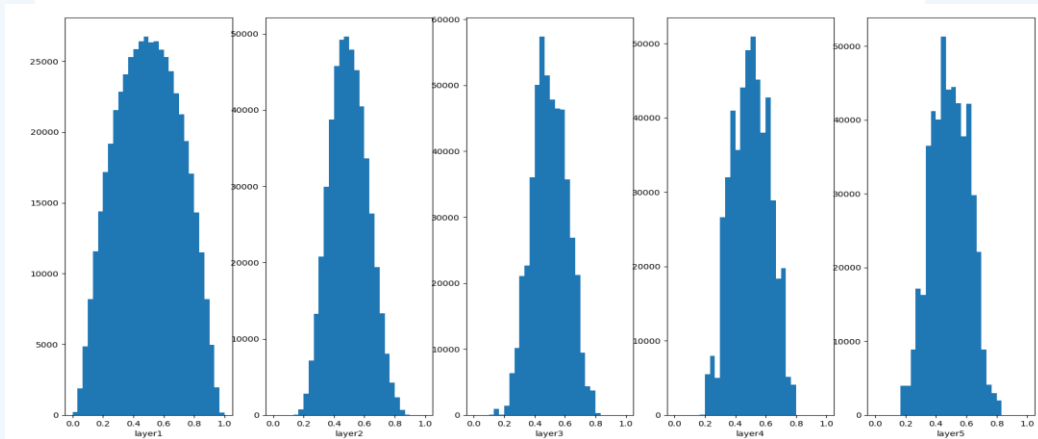


# 5. Weights Initialization (2/2): Xavier, He initialization

- **Xavier initialization**

- Useful for Sigmoid and TANH functions
- Initialize weights with the normal distribution with standard deviation of  $\frac{1}{\sqrt{n}}$  where  $n$  is the number of nodes from previous layer. n=이전 레이어의 노드 갯수

Distribution of node values at each layer

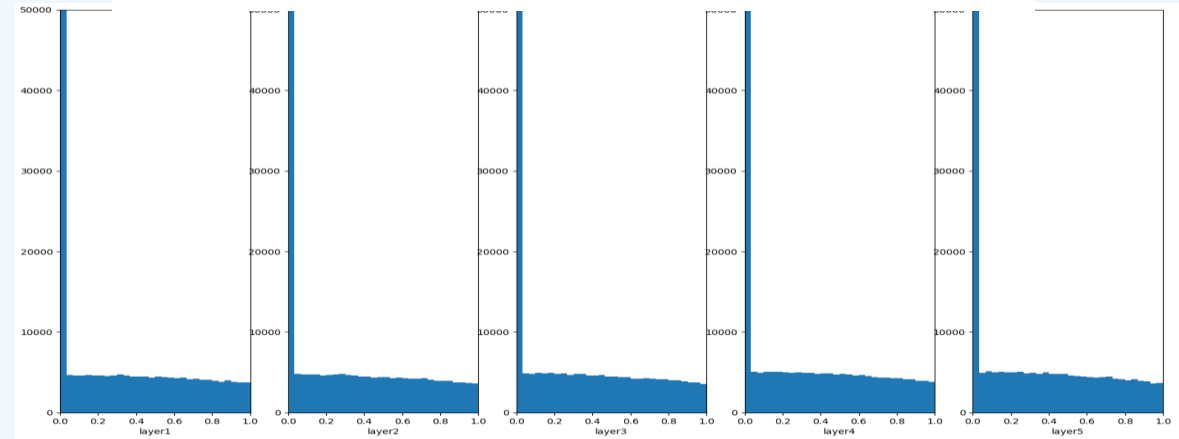


<Xavier + Sigmoid>

- **He initialization**

- Useful for ReLU function
- Initialize weights with the normal distribution with standard deviation of  $\frac{2}{\sqrt{n}}$  where  $n$  is the number of nodes from previous layer.

Distribution of node values at each layer

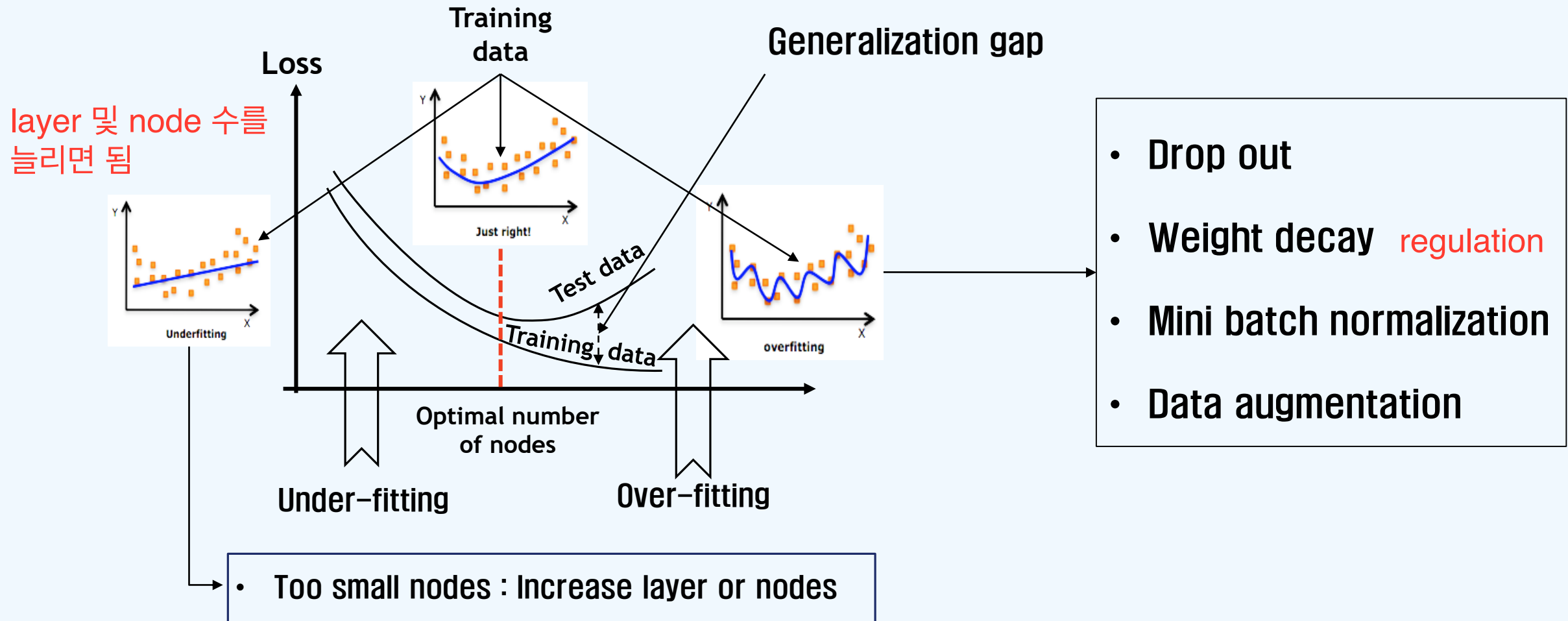


<He + ReLU>

둘다 weight값들이 한쪽에 모이지 않게 해야 함



## 6. Over-fitting vs Under-fitting (1/3) : Overview



## 6. Preventing Over-fitting (2/3) : Regulation, Dropout

- **Weight decay (Regulation)**
  - Prevent the large differences among weight values during learning process.
  - Assign penalty to large weight values
  - Add the weight penalty to loss function

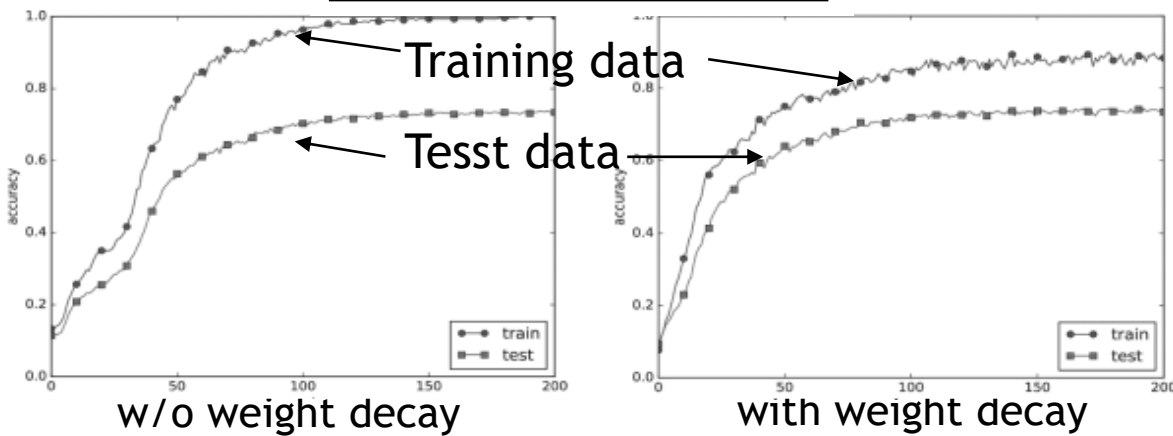
$$Loss = Loss_{data} + \lambda \|W\|^2$$



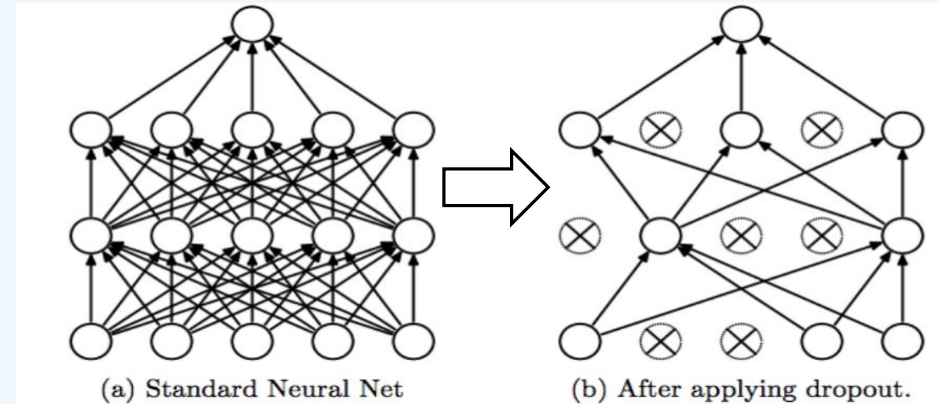
Take off some value whenever updating the weights.

$$W^{(n+1)} = (1 - \lambda \cdot \eta) W^{(n)} - \lambda \frac{\partial L}{\partial W}$$

MNIST letter classification

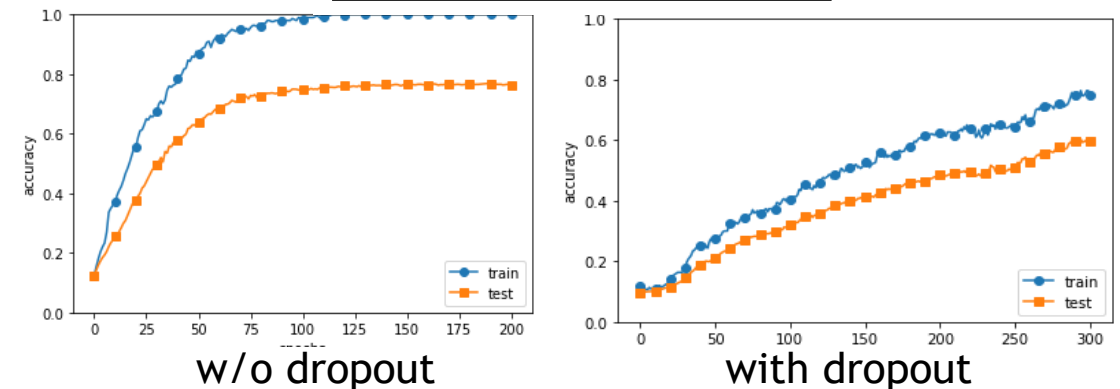


- **Dropout**
  - Randomly dropping out units (both hidden and visible)



- Repeat dropping at every training iteration
- After learning, a network uses the full connection.

MNIST letter classification





# 6. Preventing Over-fitting (3/3) : Data Augmentation

- Purpose :

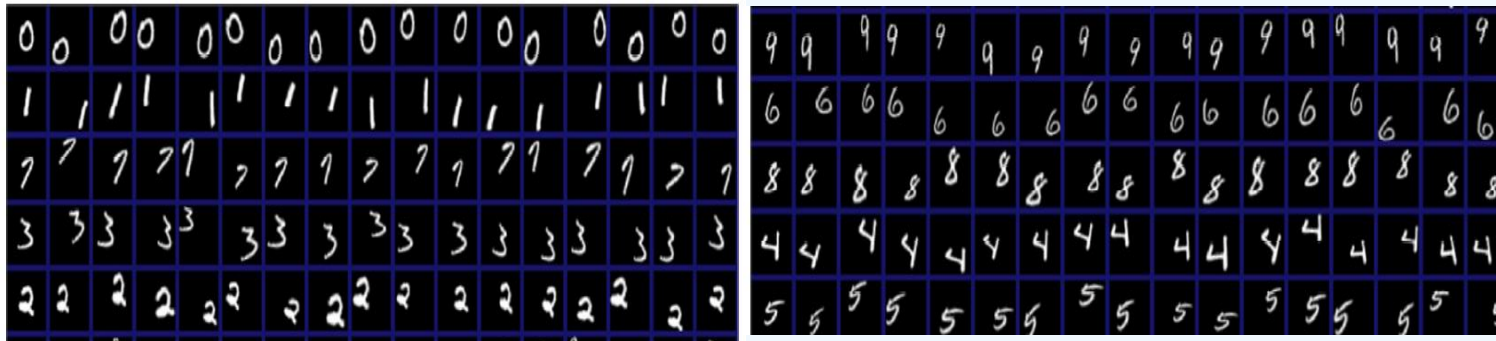
- Increasing data size => Small data size causes over-fitting problem.
- Balancing data distribution => Biased data distribution causes under-fitting problem

Data가 많으면 됨  
따라서 Data를 변형시킴

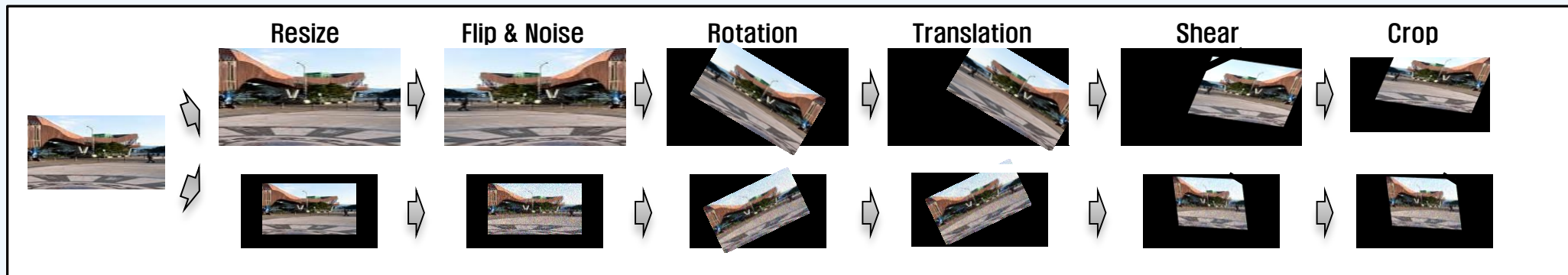
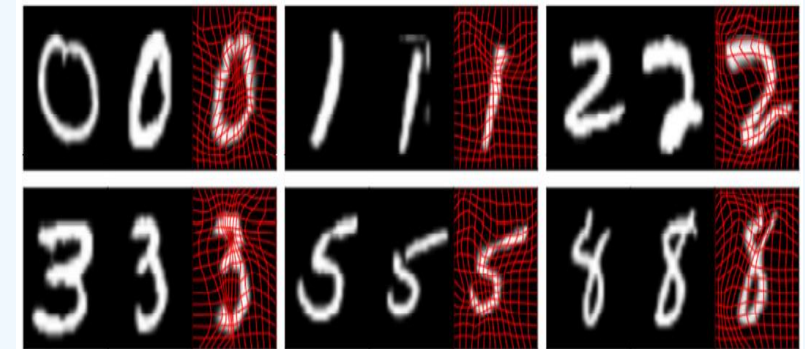
- Method: Resize&Crop, Noise Adding, Flip, Translation, Shear, Morphing...

- Example

- Transition, Rotation, Resize



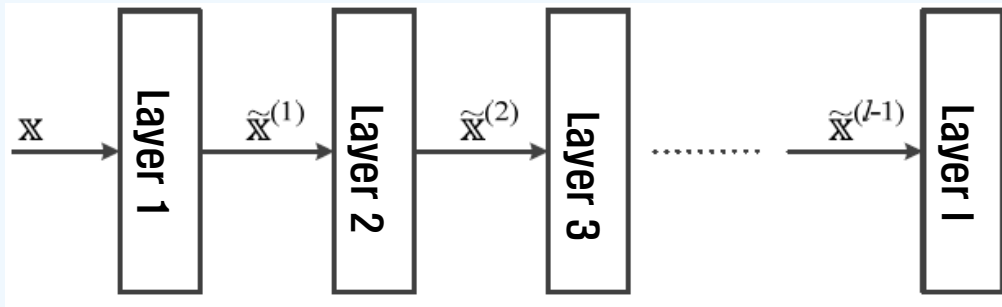
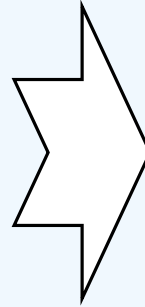
- Morphing





# 7. Batch Normalization (1/3)

- Covariance Shift
  - As learning progresses, the parameters of layers change. So, the data distributions to layers change at every layer.
  - Changes of data distribution gets serious as layer becomes deeper.
  - Critical factor preventing the learning.



각 layer마다 normalization을 해줘야 함

- To alleviate covariance shift, normalizations to all layers should be applied.
- Batch normalization is to force inputs to activation functions at each layer to have zero means/unit variances.

$$\begin{aligned} z &= \mathbf{w}^T \tilde{\mathbf{x}} + b \\ y &= \tau(z) \end{aligned}$$

$$\text{normalization : } z_i^{\text{new}} = \frac{z_i^{\text{old}} - \mu_i}{\sigma_i}$$

- Code 1 : Conduct in mini-batch
  - Code 2 : Conduct for all training set.
- Advantages of Batch normalization :
  - Large learning rate is possible. So, fast learning is possible.
  - Insensitive to initial weights
  - Regularization effect (Prevent overfitting)

# 7. Batch Normalization (2/3)

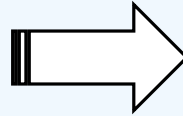
- Do code 1 with mini-batch. Then, do code 2 with training set.
- Code 1: For m batch size data  $\mathbb{X}_B = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ , the node (or input) value set  $\tilde{\mathbb{X}}_B = \{z_1, z_2, \dots, z_m\}$  is obtained as:
- Code 2: For training data set  $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ , the node (or input) value set  $\tilde{\mathbb{X}} = \{z_1, z_2, \dots, z_n\}$  is obtained as:

$$\mu_B = \frac{1}{m} \sum_{i=1}^m z_i, \quad \sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (z_i - \mu_B)^2$$

$$\tilde{z}_i = \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \quad i = 1, 2, \dots, m$$

$$z'_i = \gamma \tilde{z}_i + \beta, \quad i = 1, 2, \dots, m$$

- $\varepsilon$  : very small value, for prevent dividing by zero
- $\gamma, \beta$  : initial value is  $\gamma = 1, \beta = 0$ .  
but be updated by backward propagation.



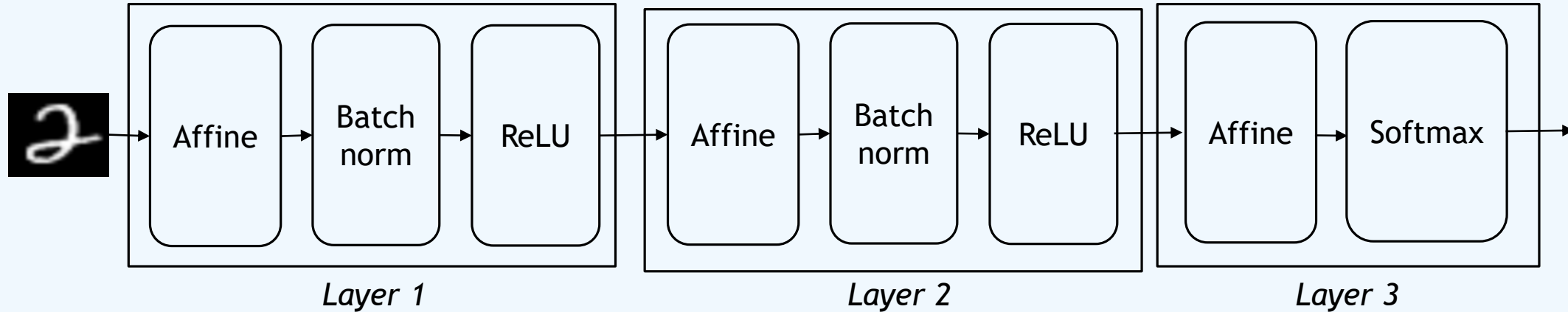
$$\mu = \frac{1}{n} \sum_{i=1}^n z_i, \quad \sigma^2 = \frac{1}{n} \sum_{i=1}^n (z_i - \mu)^2$$

- Save  $\mu, \sigma^2, \gamma, \beta$  at each node.
- In estimation step, do the forward calculation, at each node,

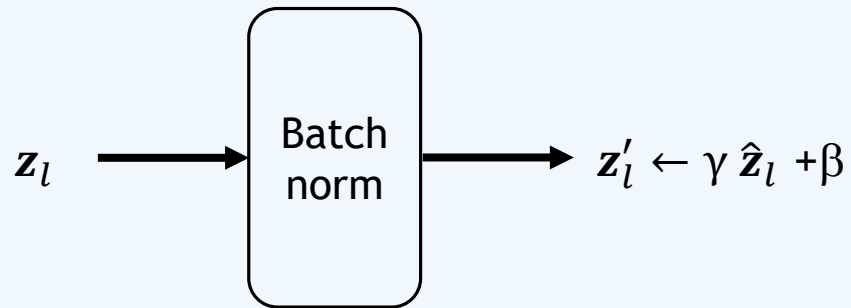
$$z' = \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} z + \left( \beta - \frac{\gamma \mu}{\sqrt{\sigma^2 + \varepsilon}} \right)$$

# 7. Batch Normalization (3/3)

- Example of a neural network using batch normalization



deep learning할 때 option들



- MNIST Classification accuracy

