Deep Learning Techniques

ML Lab

Machine
Learning
Laboratory
POSTECH

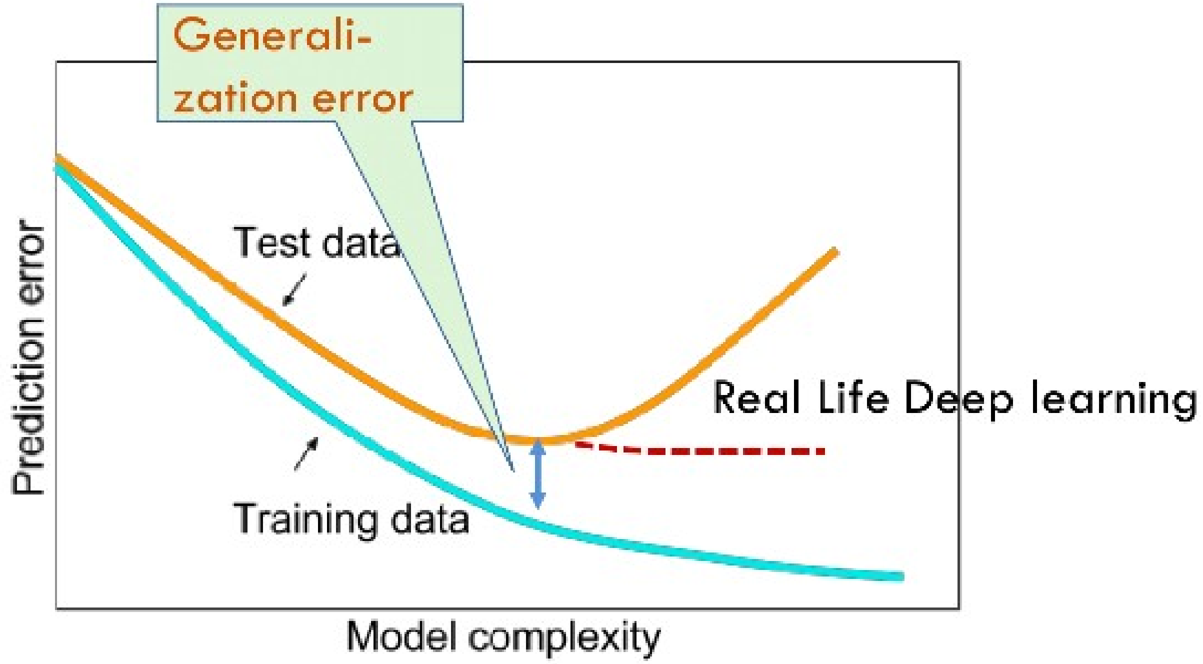
Table of Contents

Table of Contents

Training / Test error and Generalization

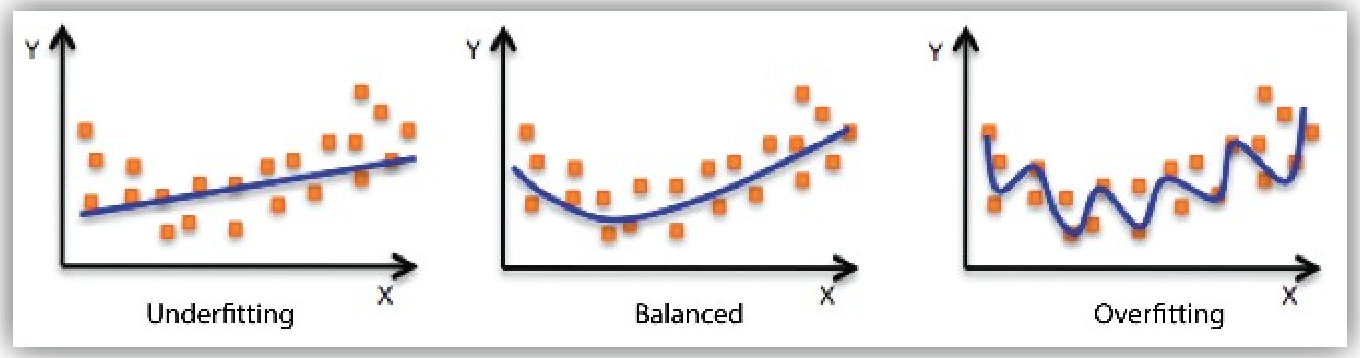
- **Training error**
  - error of training dataset
- **Test error**
  - Error of test dataset (new and previously unseen data)
- **Generalization**
  - The ability to perform well on previously unobserved input

Training/Test error and Generalization



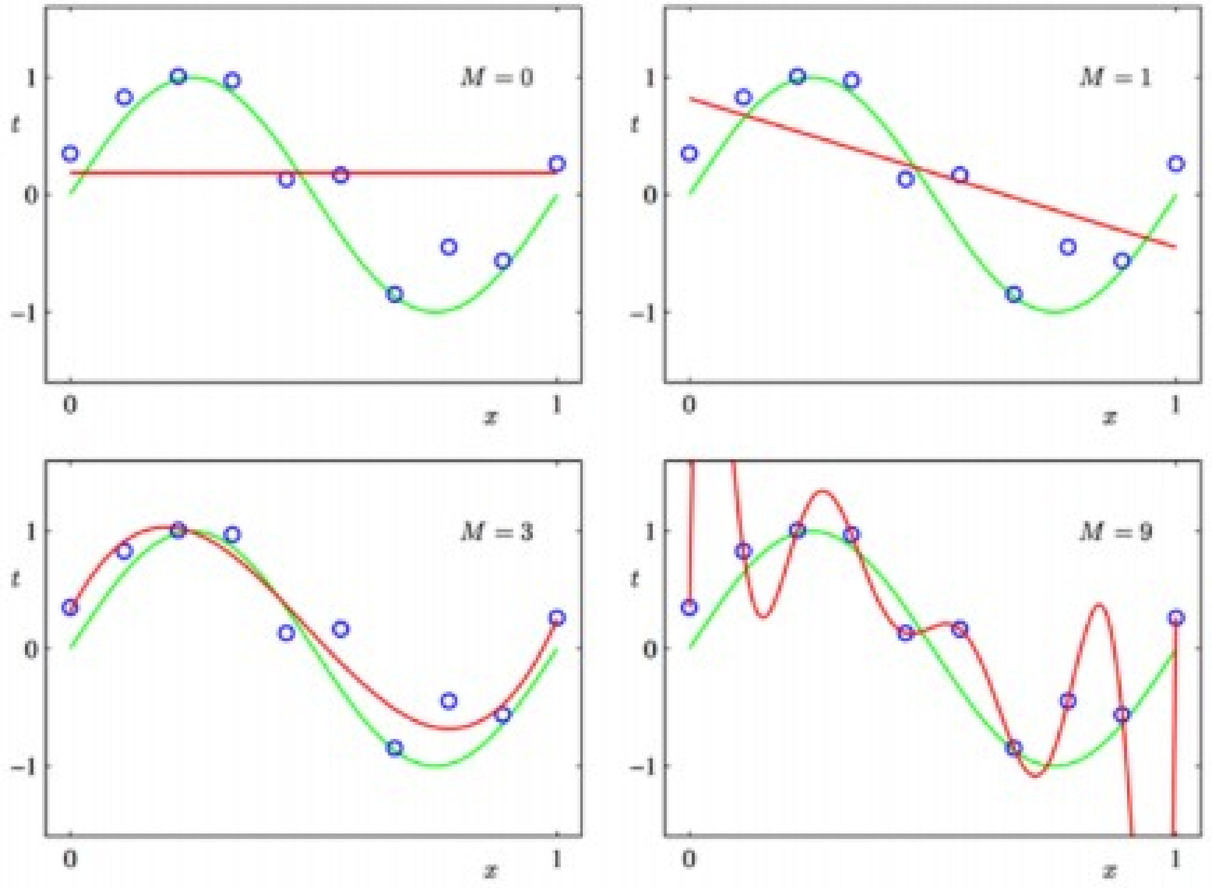
Overfitting and Underfitting

- Underfitting
  - The training data and test data have high error rates.
- Overfitting
  - The training data has a low error rate but the test data has a high error rate.

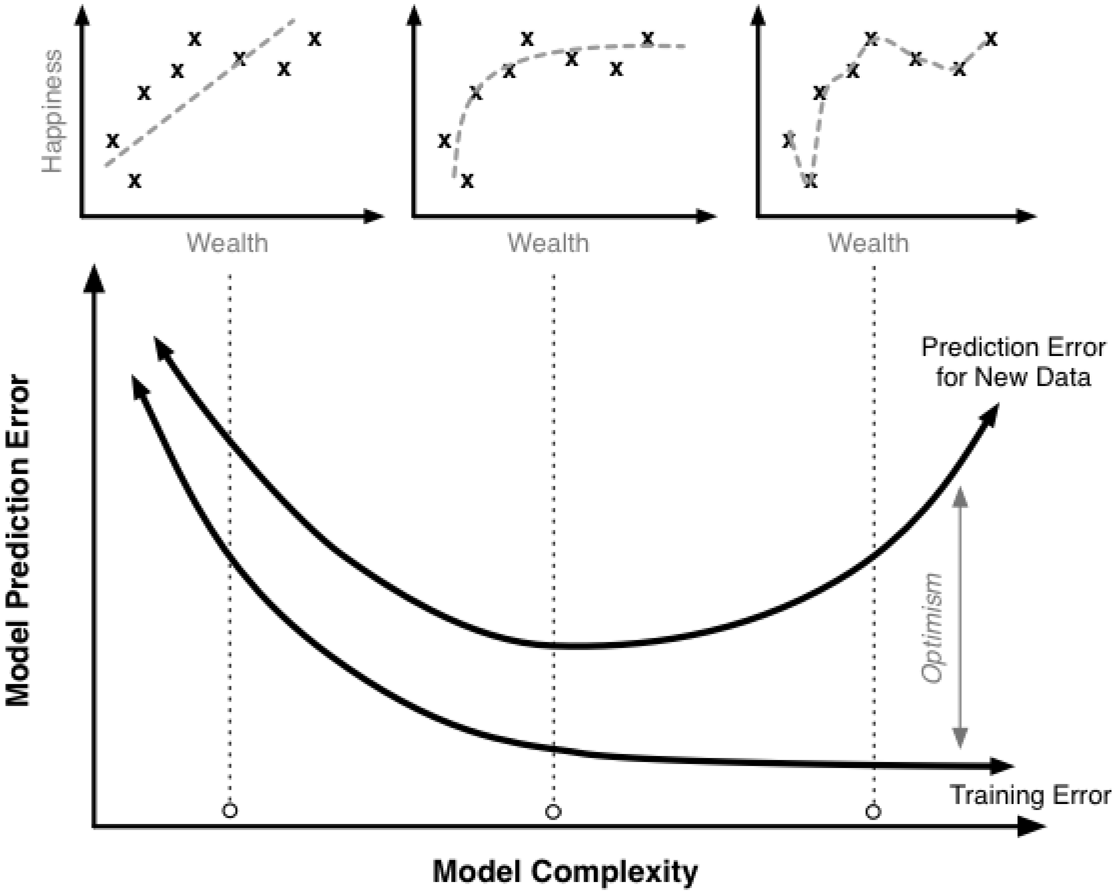


POSTECH

Overfitting and Underfitting

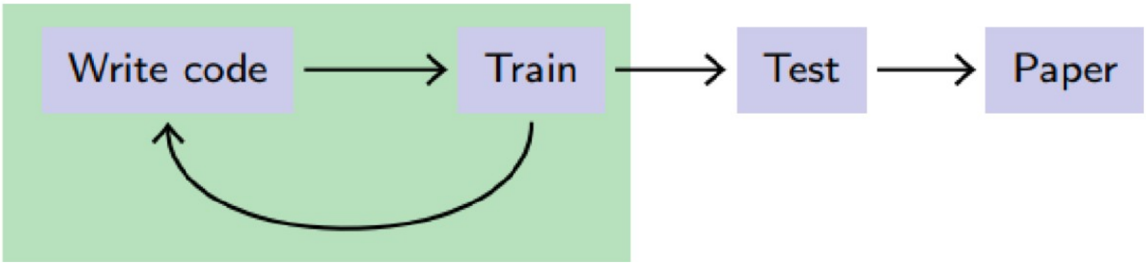


Overfitting and Underfitting



Necessity of Validation Set

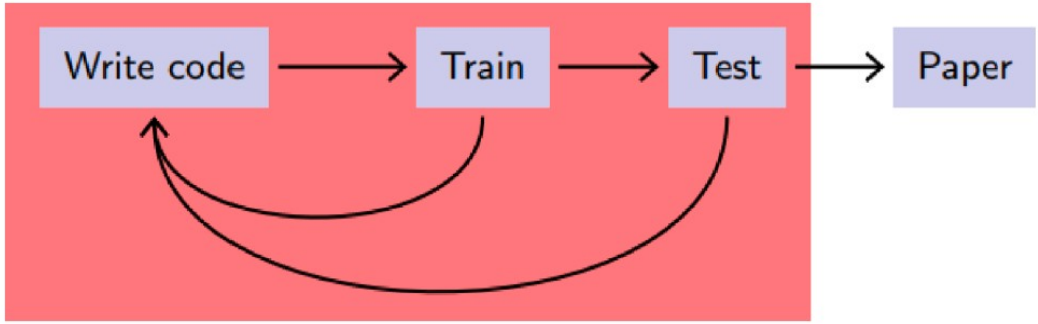
- Evaluation protocol
  - Simple training and evaluation



  - Problem: 모델에서 overfitting 문제가 일어날 수 있다

Necessity of Validation Set

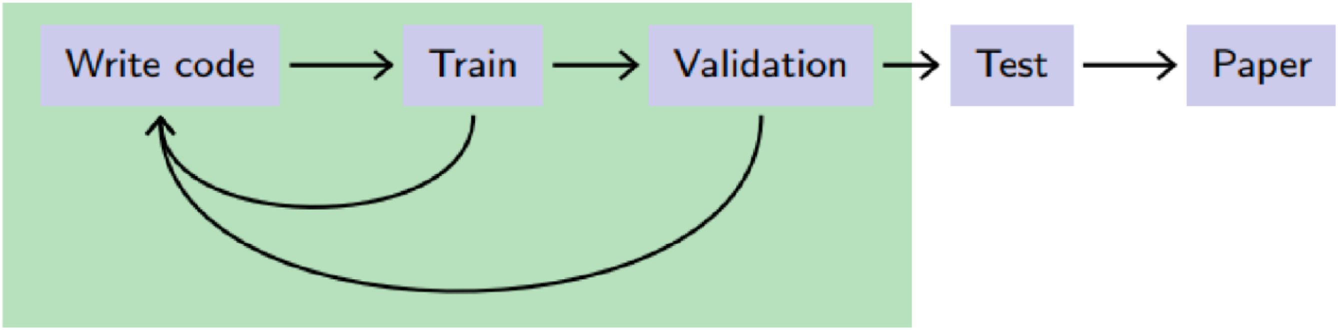
- Evaluation protocol
  - Improper training and evaluation



  - Problem: cheating!!

Necessity of Validation Set

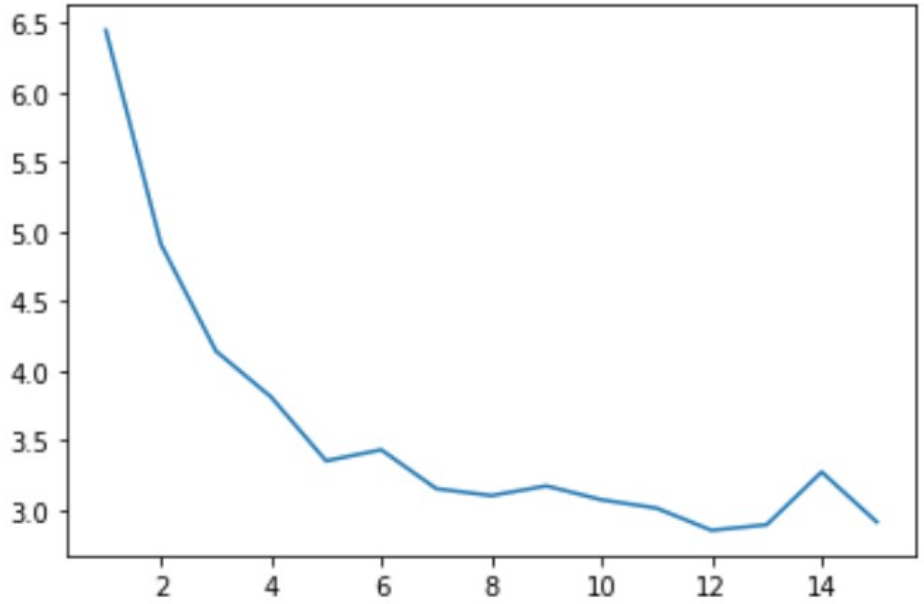
- Evaluation protocol
  - Hyper parameter 조정이나, early stopping을 위해서 validation set을 사용



  - Typically, train:validation = 4:1

Validation set

- Early stopping
  - 미리 지정한 epoch까지 다 가지 않더라도, overfitting이 일어나게 되면 학습을 중단



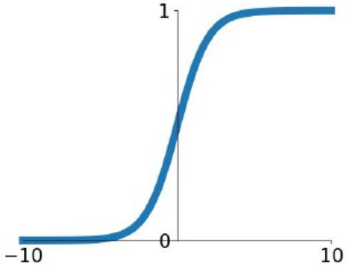
best epoch : 12

Table of Contents

Activation function

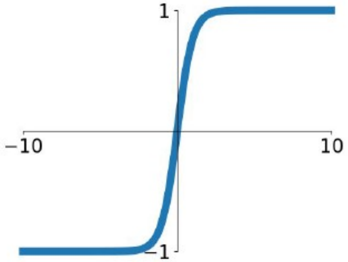
Sigmoid

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



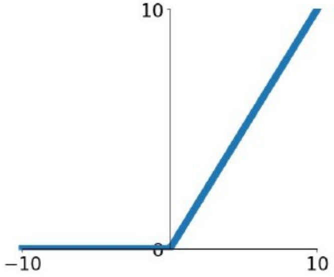
tanh

$\tanh(x)$



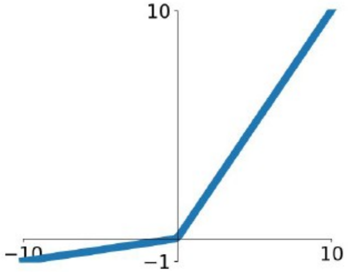
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

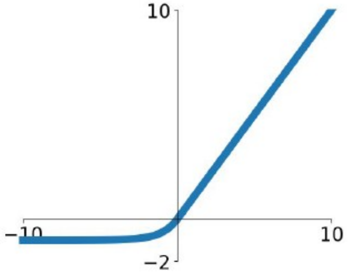


Maxout

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

ELU

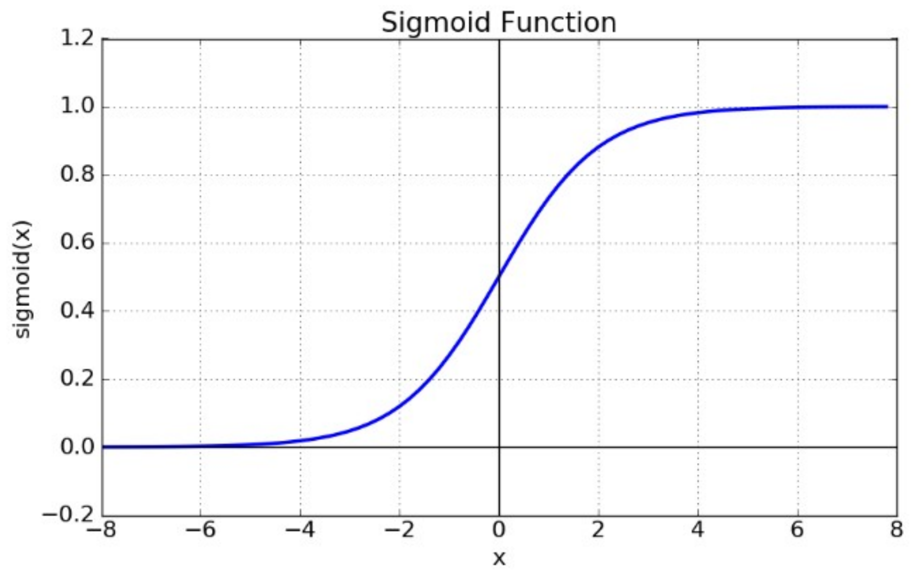
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

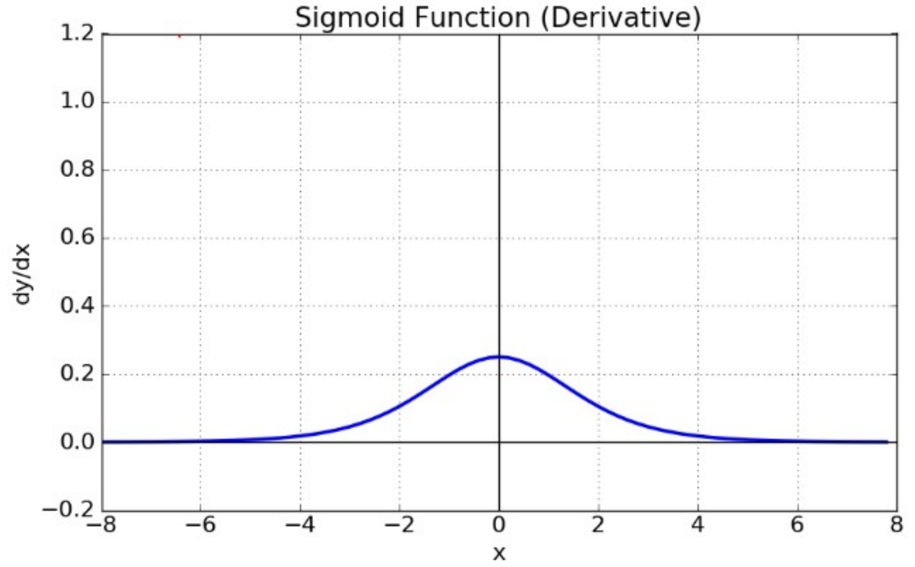


Activation function: Sigmoid

- range [0, 1]

- Problem
  - Gradient Vanishing
  - Exp() is very expensive.





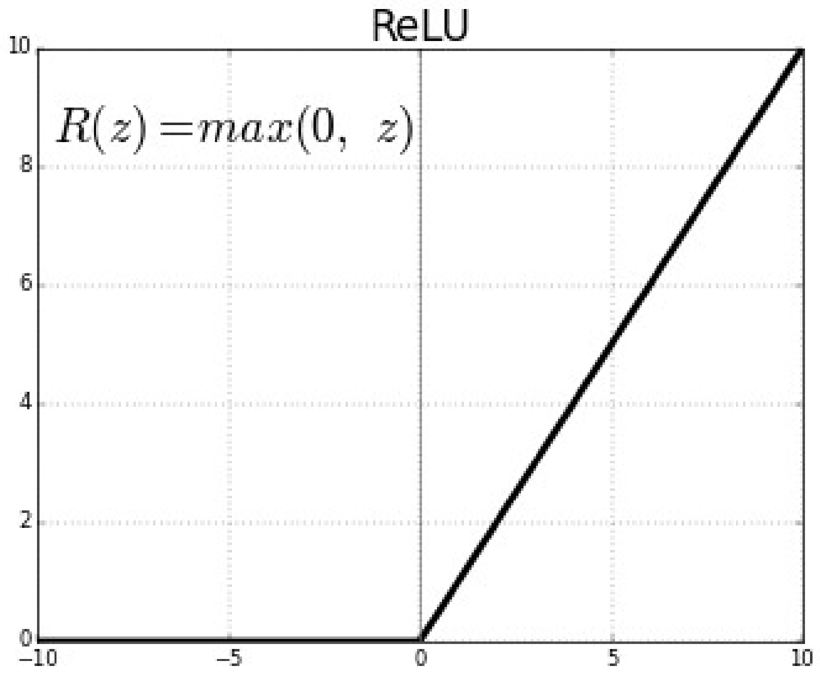
POSTECH

Activation function: ReLU

- f(x) = max(, x)

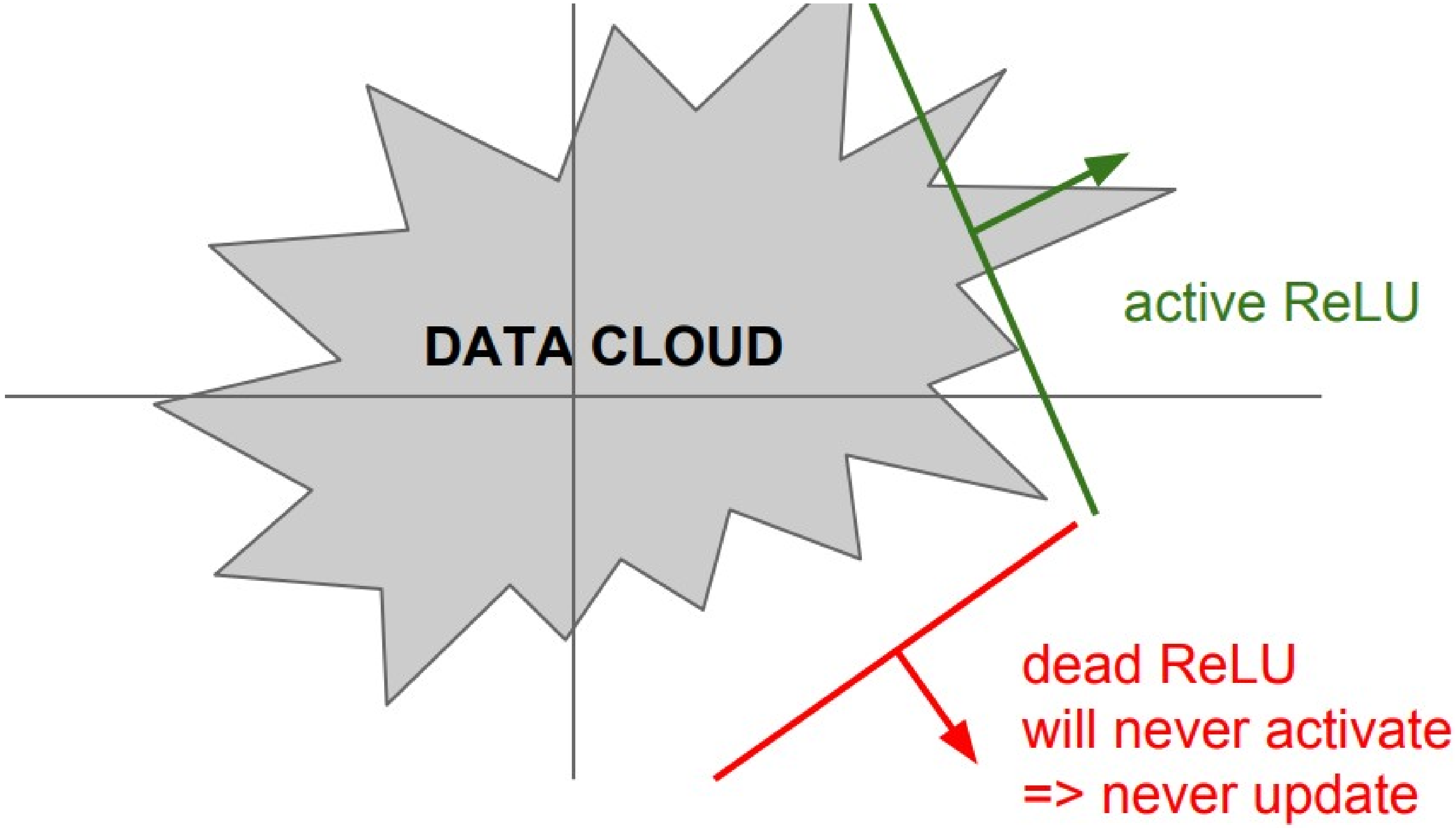
- Does not saturate (in + region)
- Very computationally efficient

- Problem
  - Dead ReLU

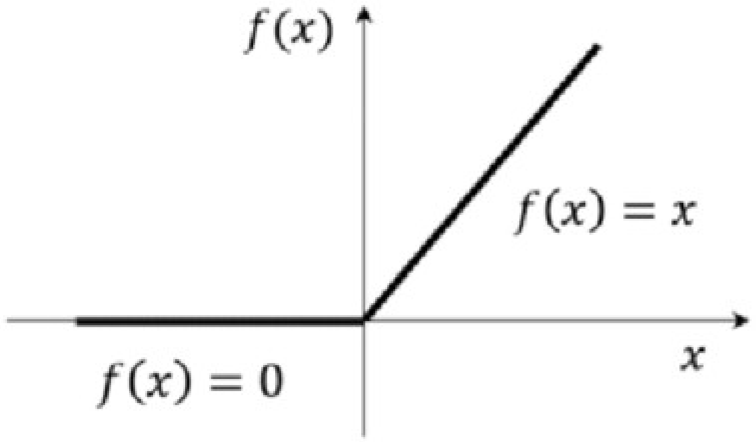


Activation function: ReLU

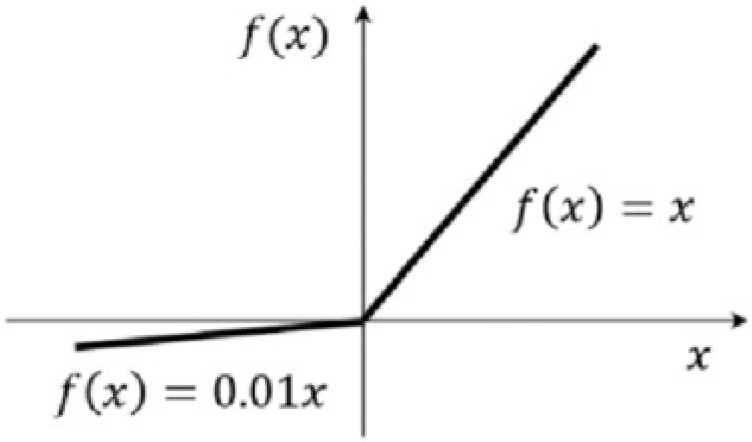
- Dead ReLU



Activation function: LeakyReLU



ReLU activation function



LeakyReLU activation function

Activation functions

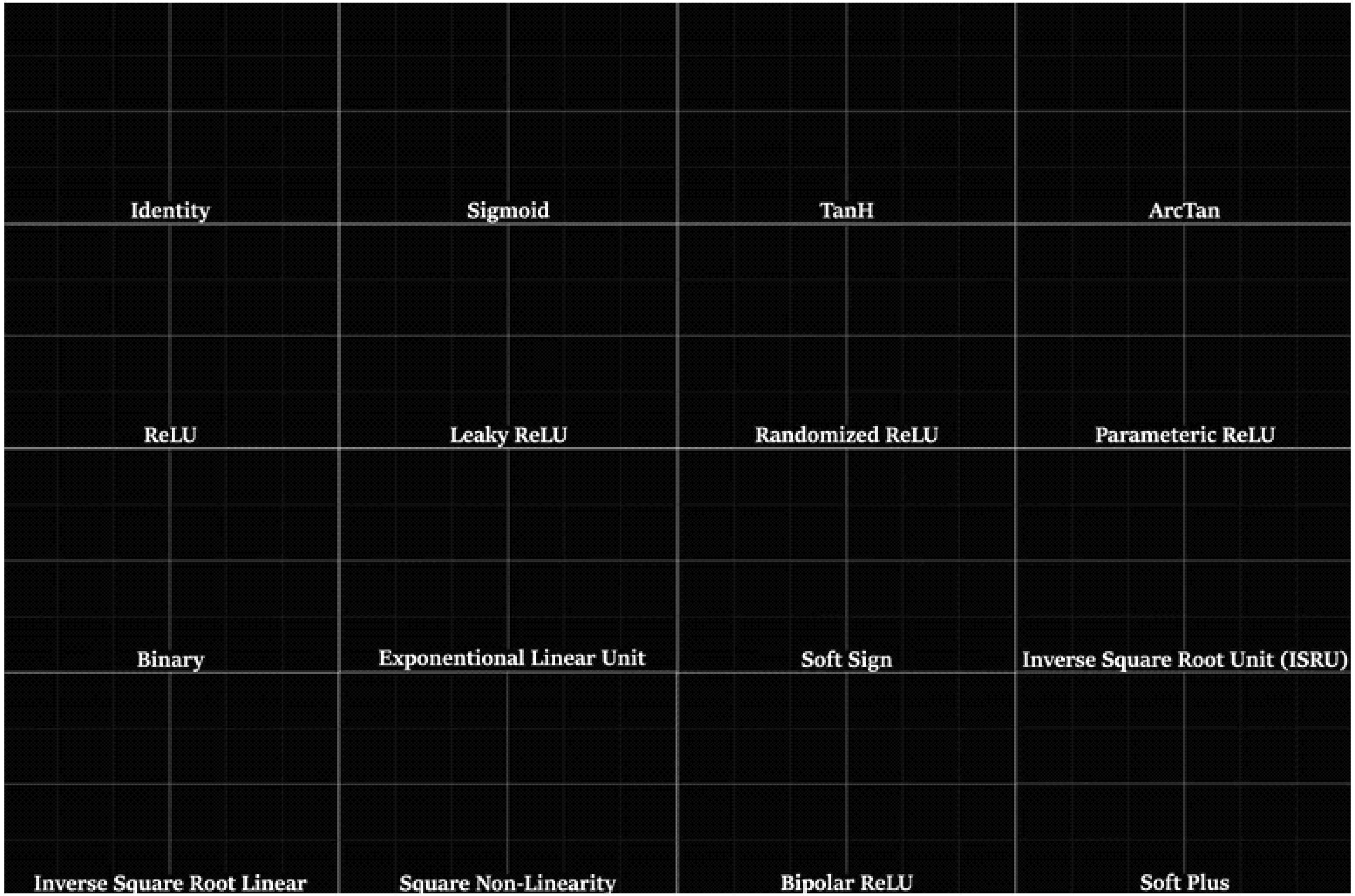
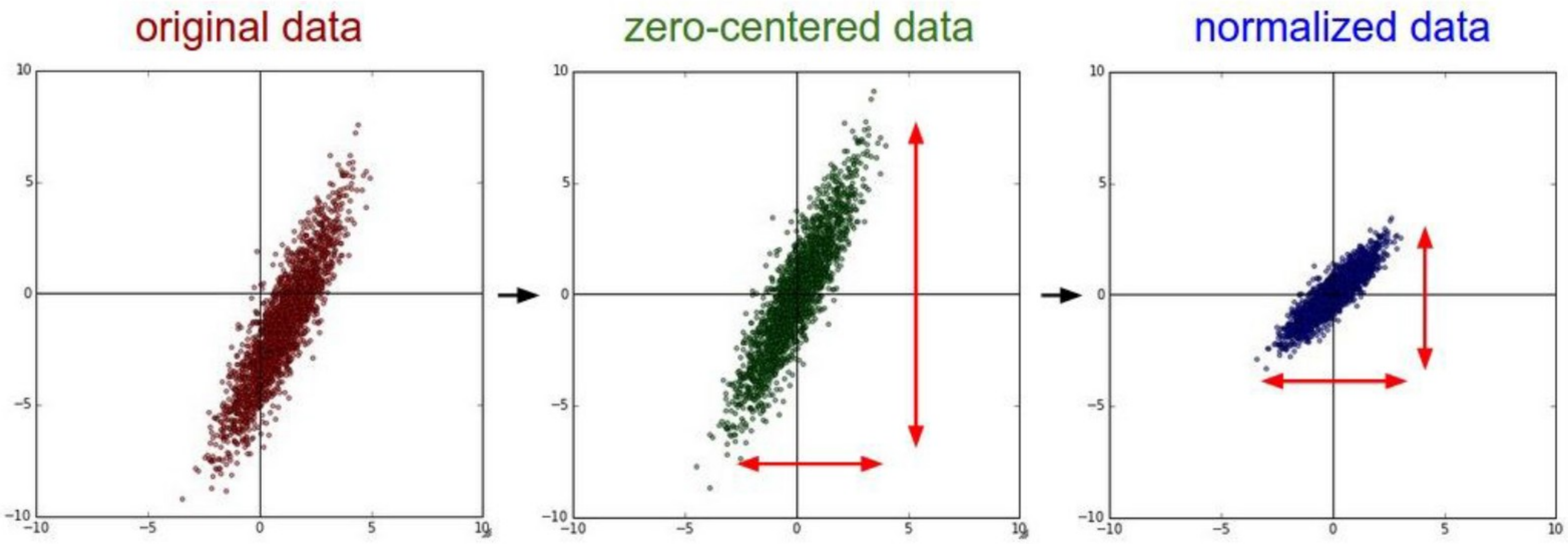


Table of Contents

Data problem: Scale

- data
  - (height(m), weight(kg))
  - ex) (1.5m, 70kg)
- The network will be biased to the 'weight'.
- To avoid this problem,
  - we must normalize the data.

Data preprocessing



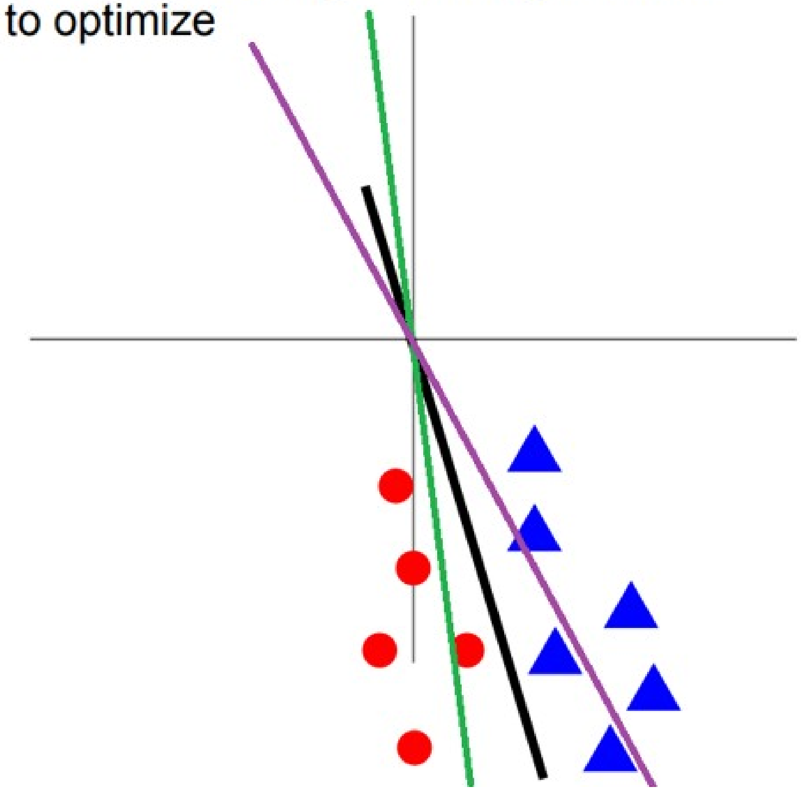
`X -= np.mean(X, axis = 0)`

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

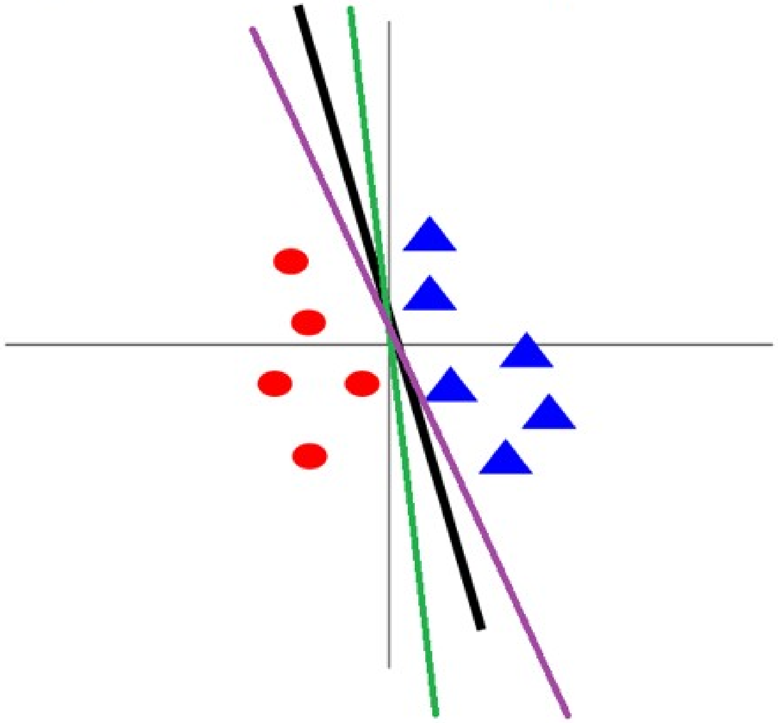
Assume X is data matrix, each sample in a row

Data preprocessing

Before normalization: classification loss very sensitive to changes in weight matrix; hard to optimize

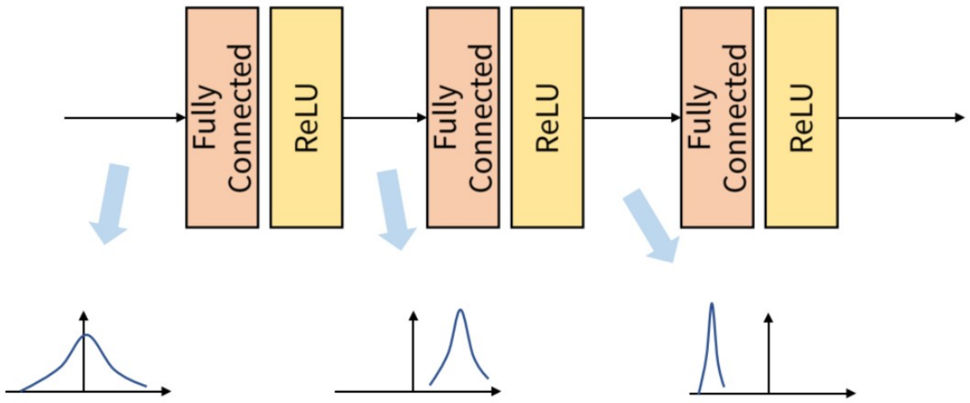


After normalization: less sensitive to small changes in weights; easier to optimize



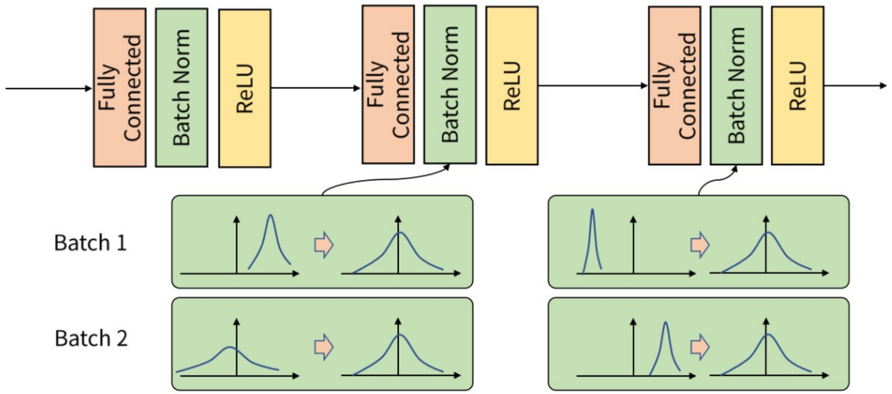
Batch normalization

- Internal covariate shift 현상의 해결
  - 학습 중간에 weight 가 업데이트 되면서 학습데이터의 분포가 계속 바뀌는 현상으로 인해 batch 간 데이터 분포의 차이가 발생
  - Batch 단위로 학습할 경우 하나의 데이터 셋을 다 사용하기 전에 update 가 일어나기 때문



Batch normalization

- 이러한 현상을 해결하기 위해 batch 및 layer 마다 정규화를 진행



  - 학습단계에서는 learning parameter 를 이용하여 평균 및 분산을 구하고 , 배치 정규화를 진행
  - Test 단계에서는 학습에서 사용했던 평균 및 분산을 사용

Batch normalization

- Input: $X \in \mathbb{R}^{N \times D}$
- What if zero-mean, unit var is too hard of a constraints?
- Learnable scale and shift parameters:

$$\gamma, \beta \in \mathbb{R}^D$$

- Learning $\gamma = \sigma, \beta = \mu$ will recover the identity function!

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$$

Per-feature mean. Shape is *D*

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{ij} - \mu_j)^2$$

Per-feature var. Shape is *D*

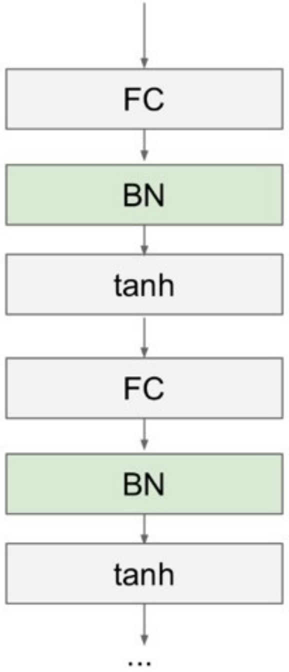
$$\hat{x}_{ij} = \frac{x_{ij} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized x. Shape is *N*×*D*

$$y_{ij} = \gamma_j \hat{x}_{ij} + \beta_j$$

Output. Shape is *N*×*D*

Batch normalization



- Makes deep networks much easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!

Other normalization techniques

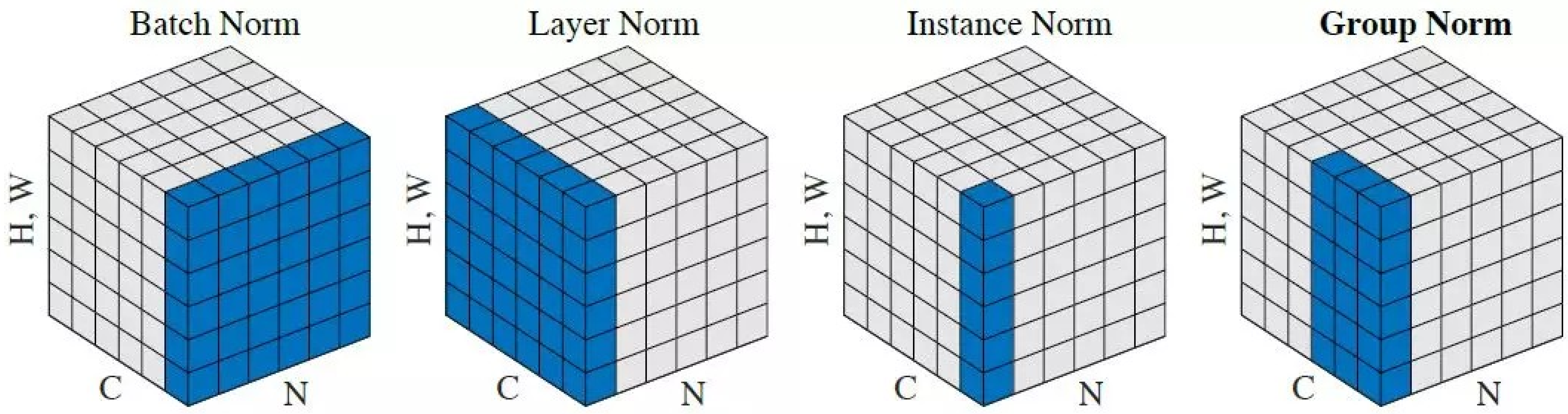


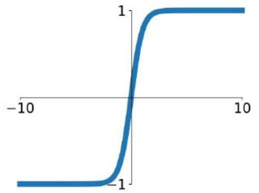
Table of Contents

Weight initialization

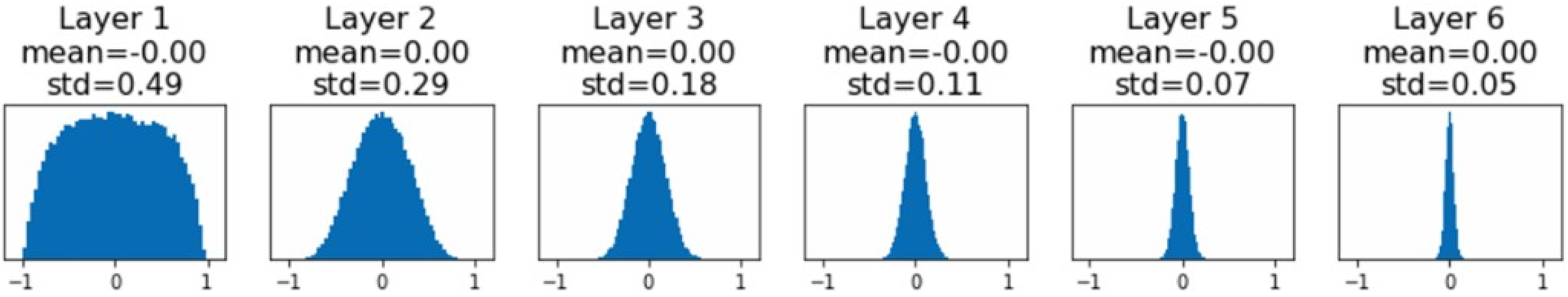
- Init weights with with **tanh(x)**,

tanh

$\tanh(x)$



- All zero, no learning!

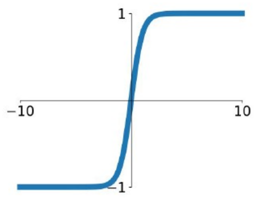


Weight initialization

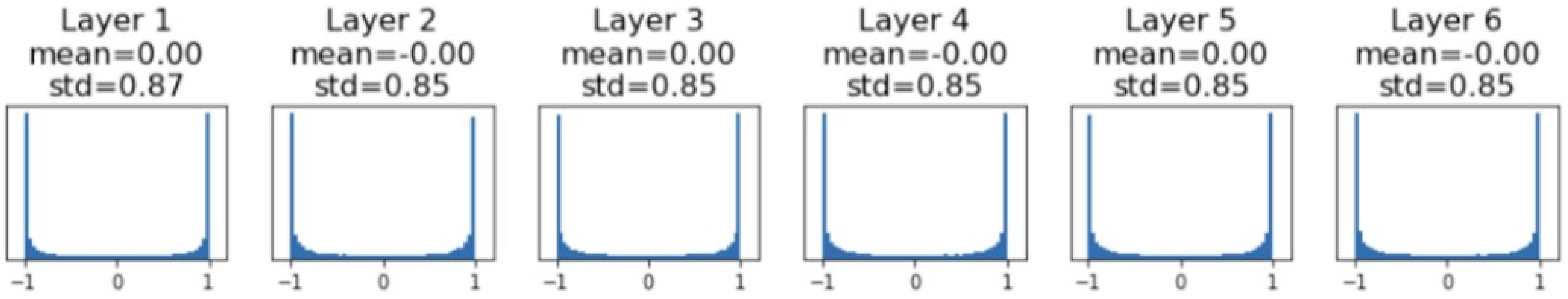
- Init weights with with **tanh(x)**,
- Local gradient all zero, no learning!

tanh

$\tanh(x)$

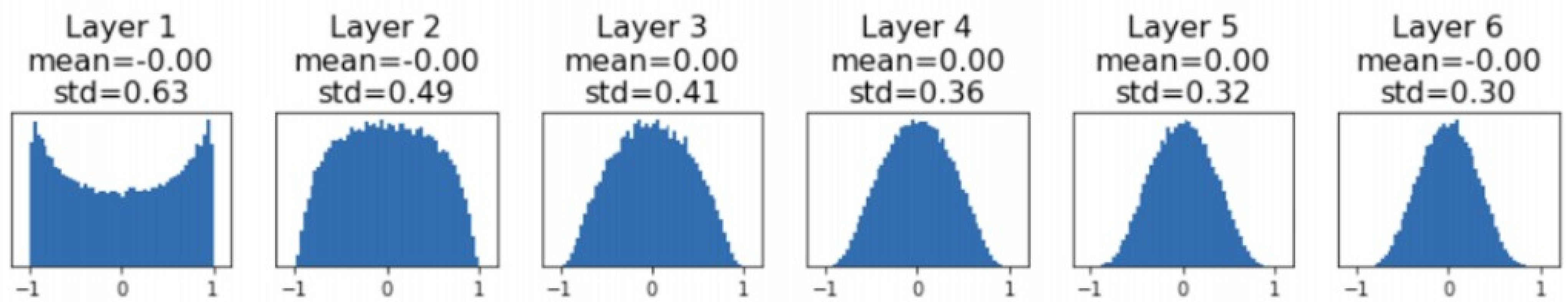


$$\frac{d}{dx}\tanh(x) = 1 - \tanh^2(x)$$



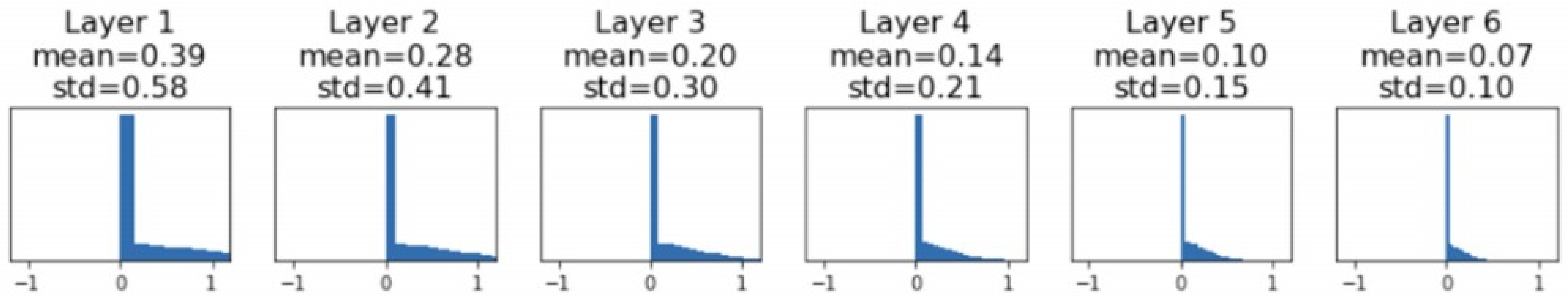
Xavier initialization

- Init weights with with **tanh(x)**,
  - Var(W) =
  - : the size of the dimension of input



Xavier initialization

- Init weights with with **ReLU(x)**,
  - Var(W) =
  - : the size of the dimension of input



He initialization

- Init weights with with **ReLU(x)**,
  - Var(W) =
  - : the size of the dimension of input

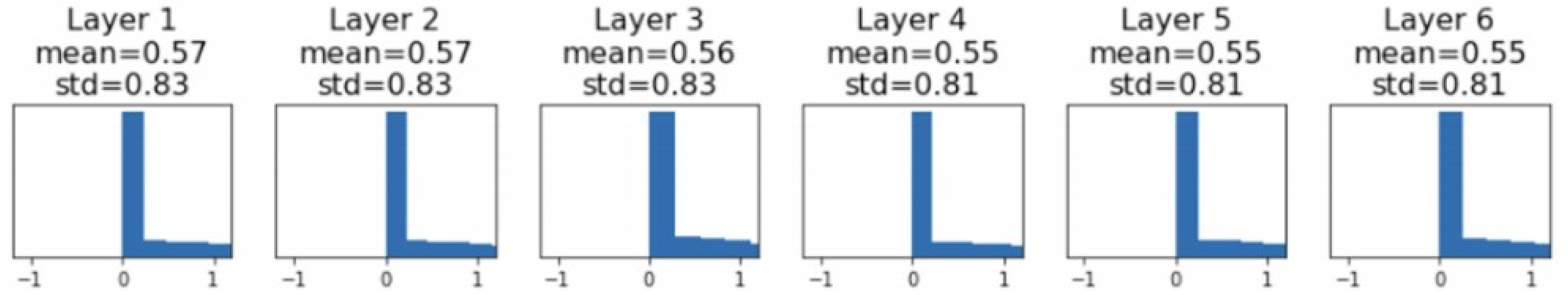


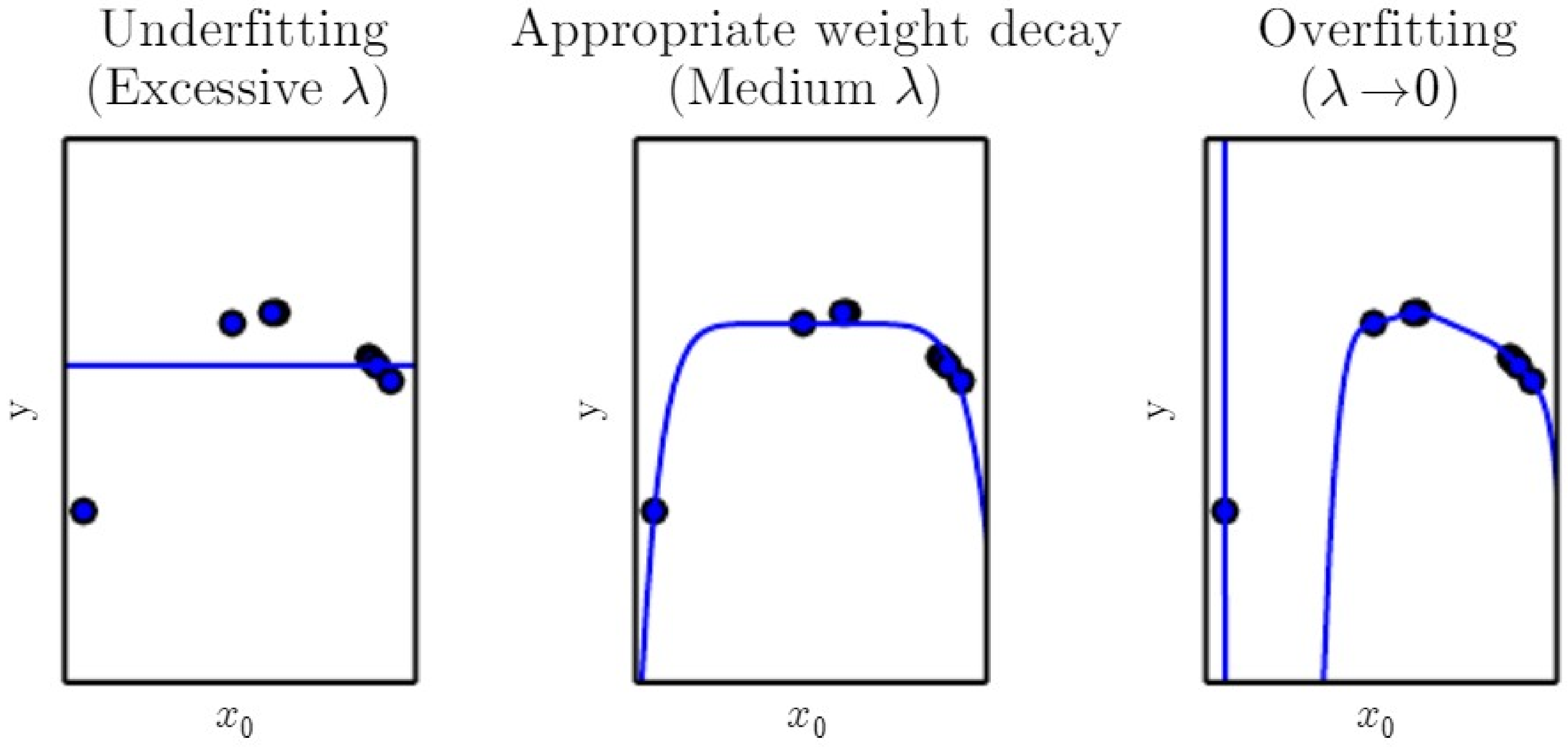
Table of Contents

Weight Decay

$$L(W) = \frac{1}{N}\sum_{i=1}^{N}\ell(y_i, f_W(x_i)) + \boxed{\lambda R(W)}$$

- In common use
  - L2 regularization $R(W) = \sum_{i,j} W_{ij}^2$
  - L2 regularization $R(W) = \sum_{i,j} |W_{ij}|$
  - Elastic net (L1 + L2) $R(W) = \sum_{i,j} \beta W_{ij}^2 + |W_{ij}|$

Weight Decay



Regularization

- Weight decay 에서는 linear function 이 작은 weight 을 갖는게 좋다는 것을 loss function 에 추가적인 term 을 추가하여 explicit 하게 표현함
- 이러한 선호도를 implicit 또는 explicit 하게 표현하는 방식으로는 많은 방법이 있고 이러한 방식을 regularization 이라고 말함
- Regularization 의 목적은 generalization error 를 줄이는 것임

Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org

Table of Contents

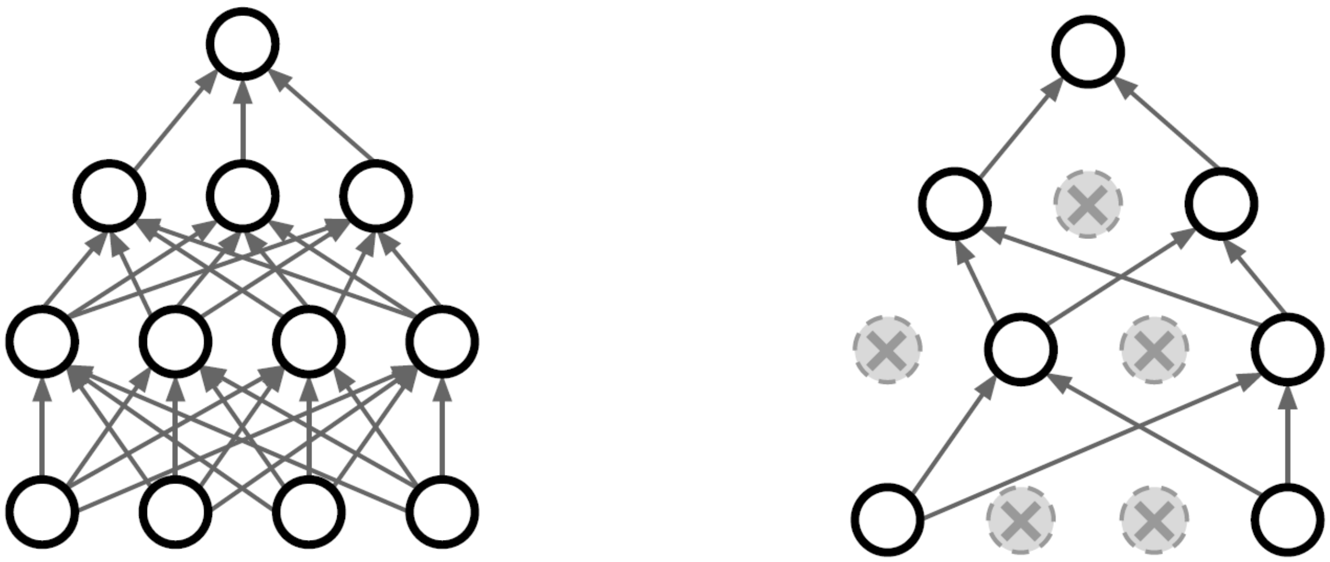
Model Ensembles

- Train multiple independent models
- At test time, average their results
  - Take average of predicted probability distributions, then choose argmax

- 여러 개의 다른 모델을 따로 학습시켜서 그 모델들이 test data 의 output 값에 기여하도록 하겠다는 아이디어
- 다른 모델들이 한가지 test set 에서 모두 똑같은 결과를 보이지는 않을 거라는 아이디어

Dropout

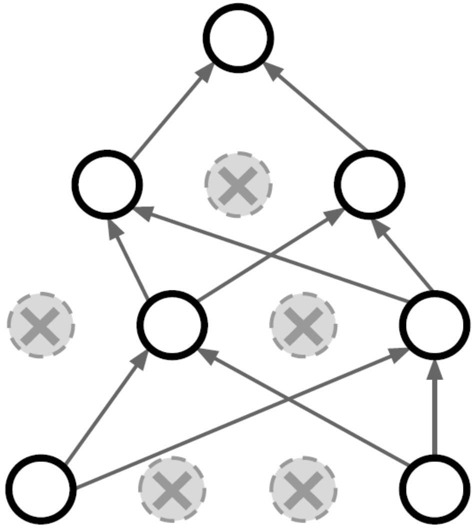
- 미리 정한 확률로 특정 뉴런을 학습에서 제외할지 정함
  - 보통 확률을 0.5를 사용함



Srivastava et al, “Dropout: A simple way to prevent neural networks from overfitting”, JMLR 2014

How to work?

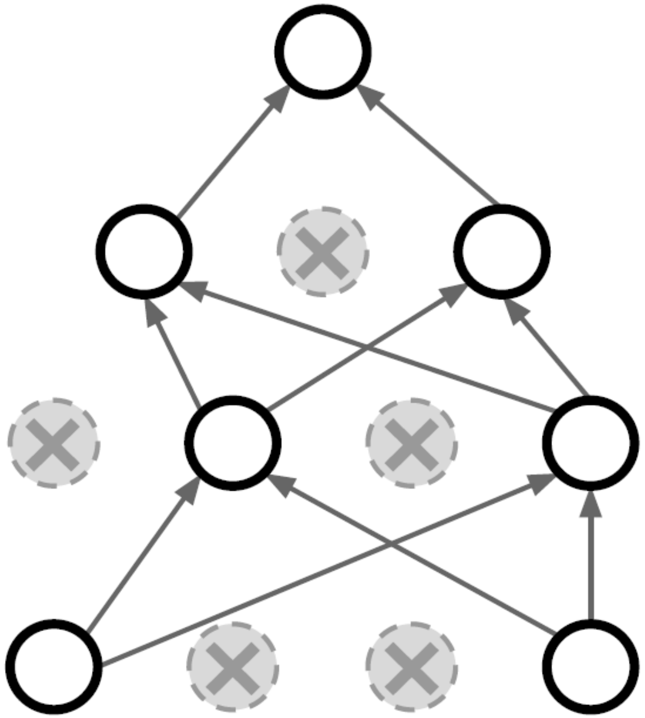
- Feature 의 co-adaptation 을 방지하여 네트워크에 불필요한 representation 을 제외한다
  - Co-adaptation 이란 한 뉴런이 다른 뉴런에 의존적으로 학습되는 현상



Forces the network to have a redundant representation;
Prevents co-adaptation of features

Another Interpretation

- Dropout 은 확률적으로 특정 뉴런을 사용하는지 결정
  - 각각의 step 마다 다른 network 를 학습시키는 것으로 볼 수 있고, 이를 model ensemble 관점에서 볼 수 있음

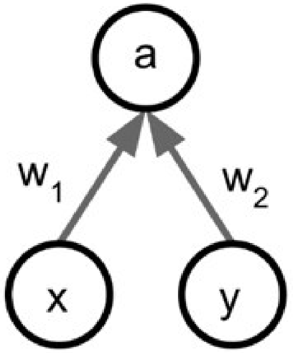


Dropout in test time

- Test time 에는 dropout 확률을 곱해줘야 함
  - Test time 에서의 output 은 train time 에서의 output 의 기댓값

- Test 에서 a 값의 기대값 :

$$E[a] = w_1x + w_2y$$

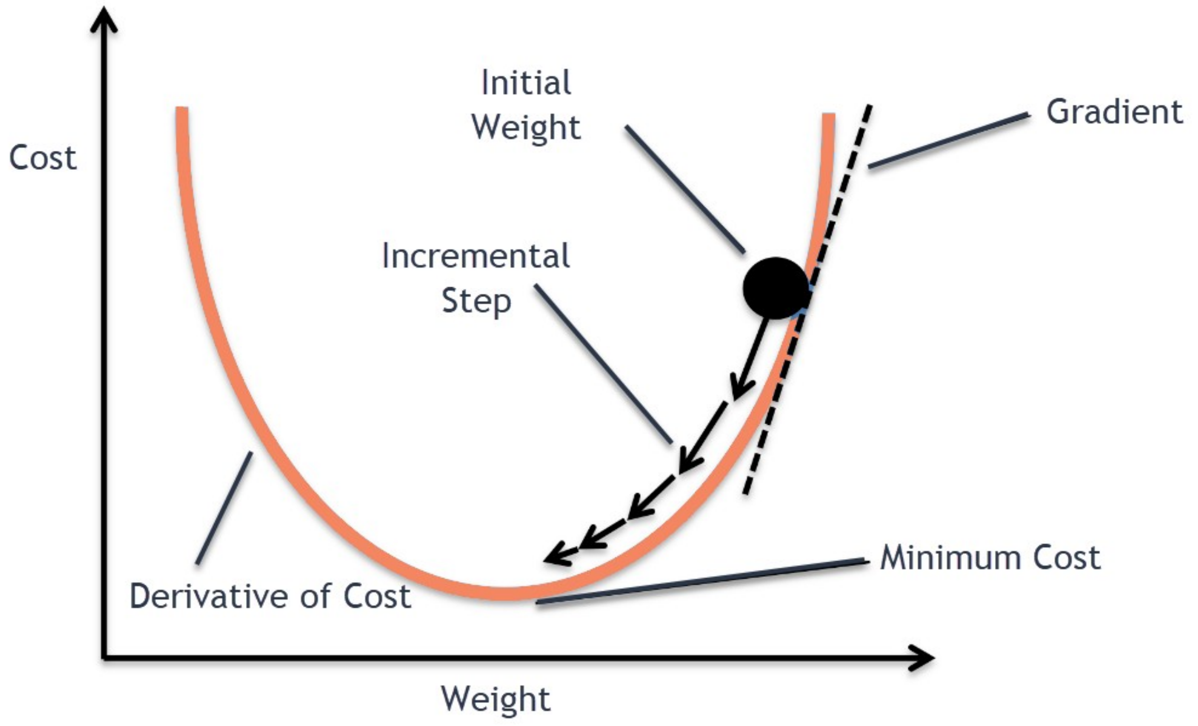


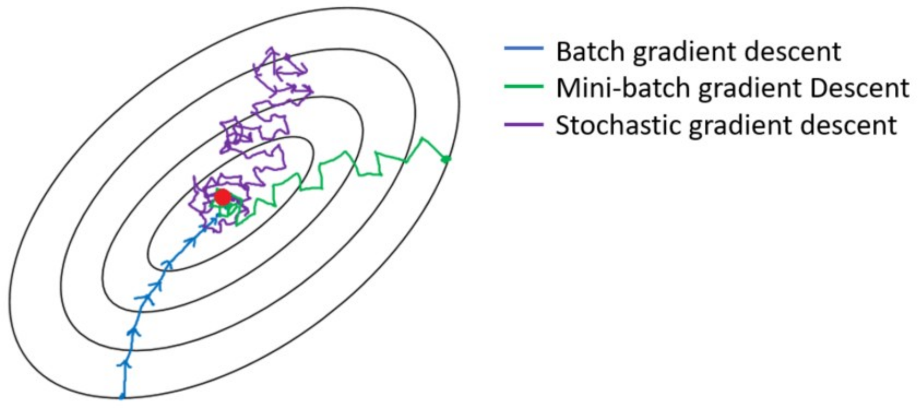
- 학습에서 a 값의 기대값 :

$$E[a] = \frac{1}{4}(w_1x + w_2y) + \frac{1}{4}(w_1x + 0y) + \frac{1}{4}(0x + 0y) + \frac{1}{4}(0x + w_2y) = \frac{1}{2}(w_1x + w_2y)$$

Table of Contents

Stochastic Gradient Descent (SGD)

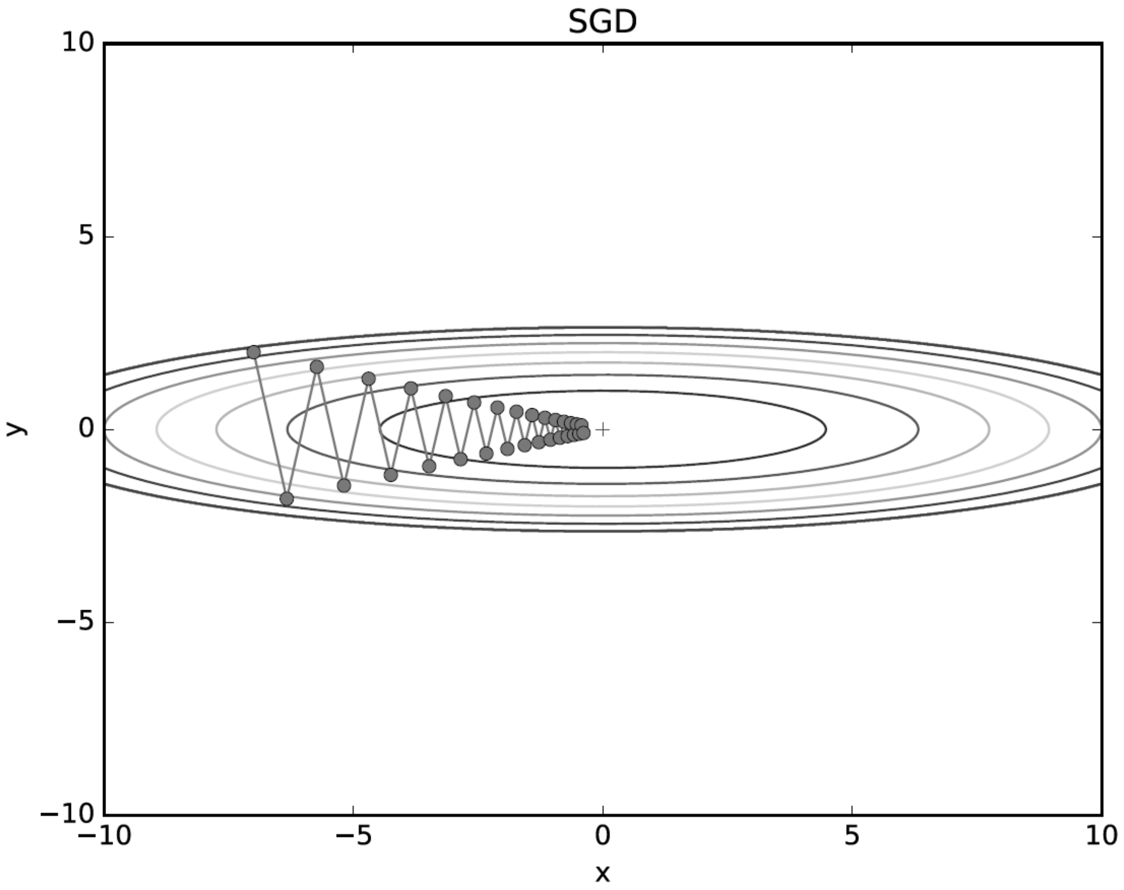




$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

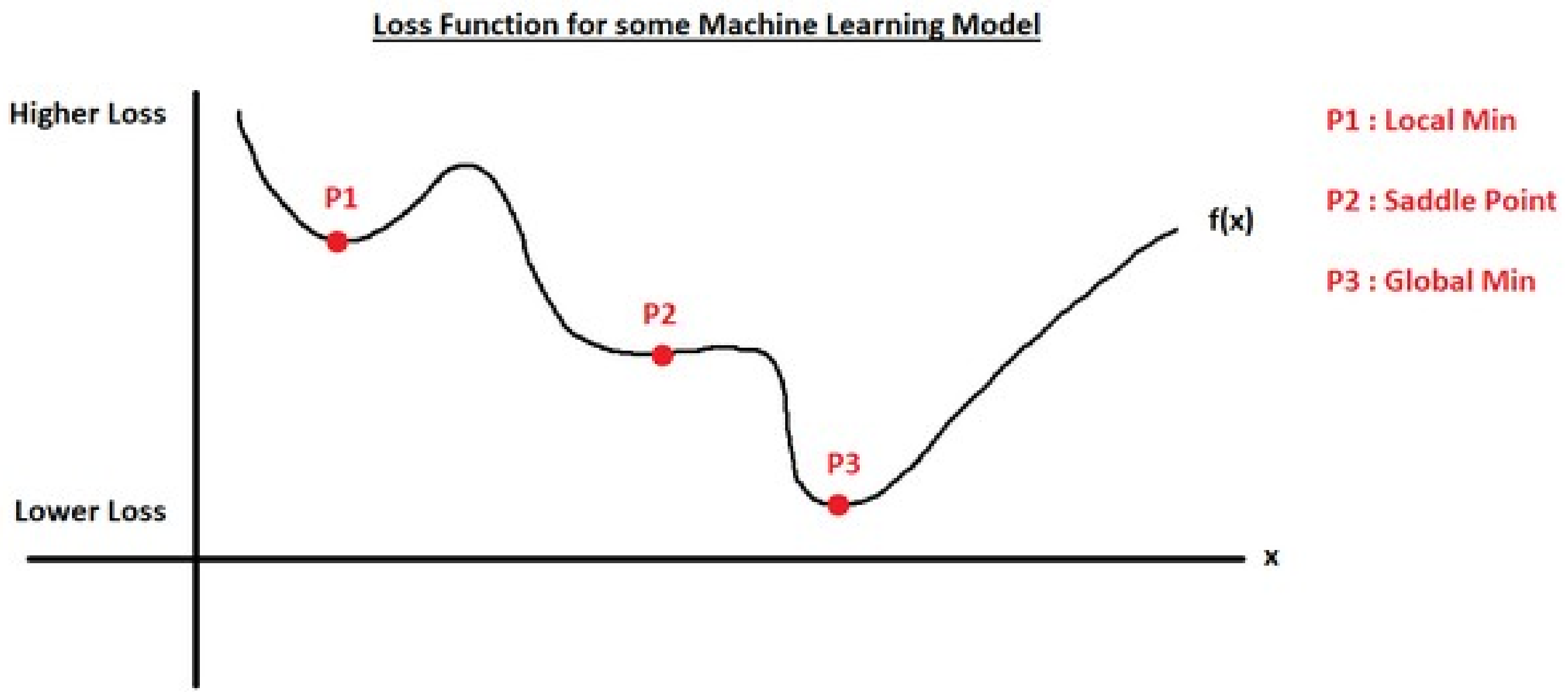
Stochastic Gradient Descent (SGD)

- Problem

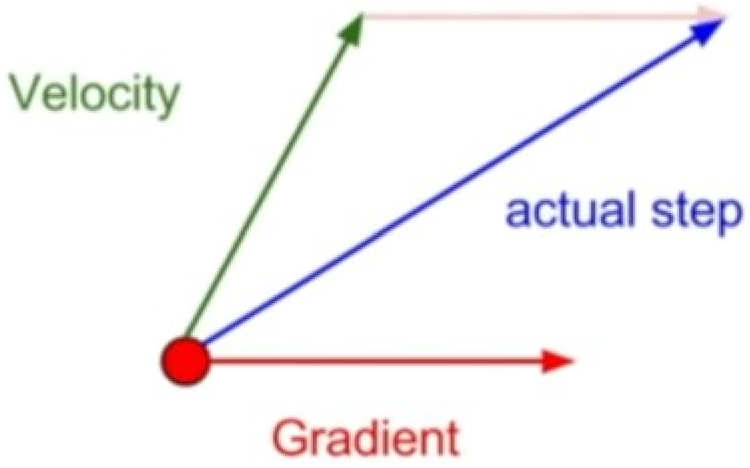


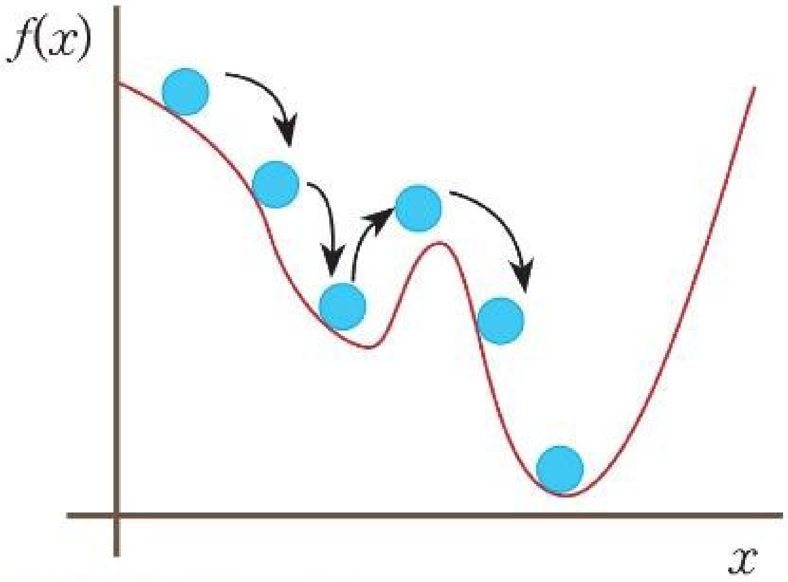
Problem of gradient descent

- Zero gradient, gradient descent gets stuck.



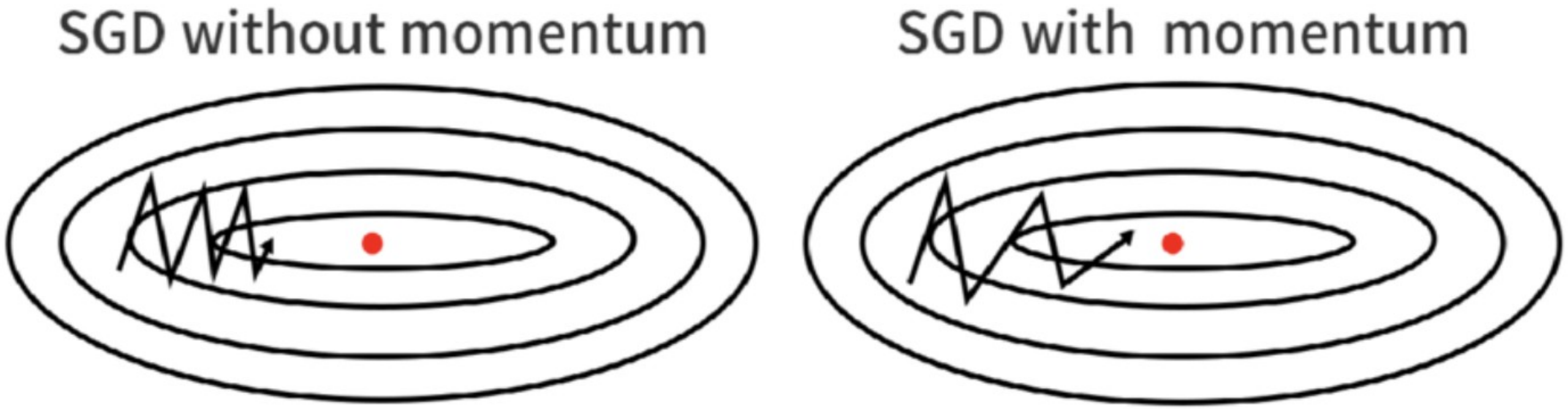
Momentum





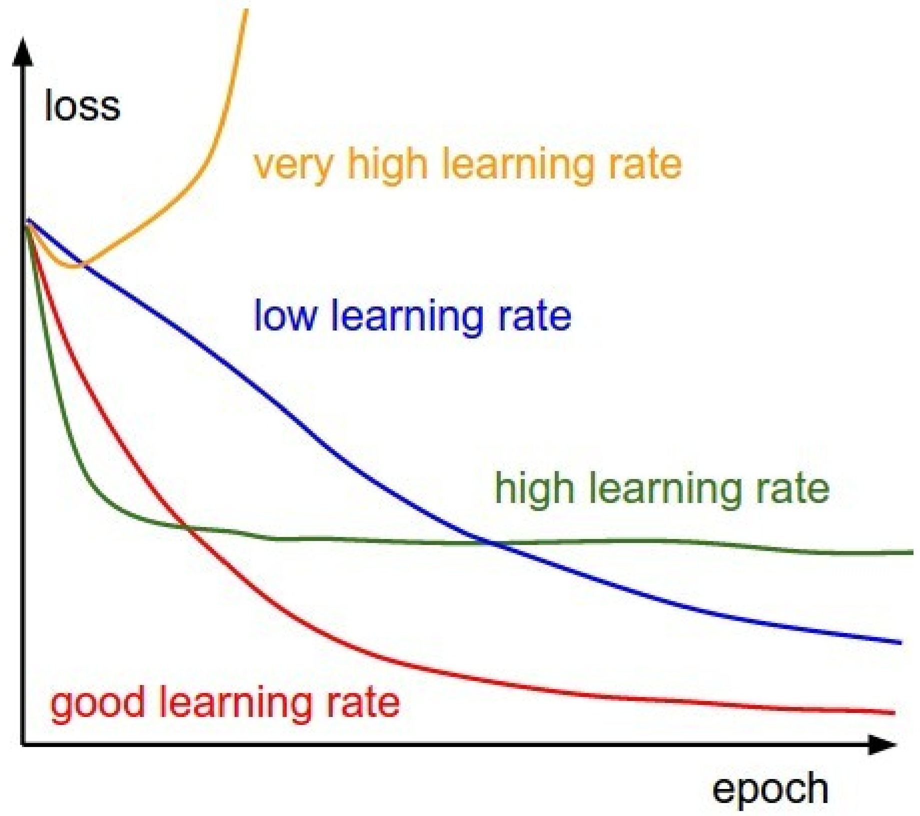
SGD + Momentum

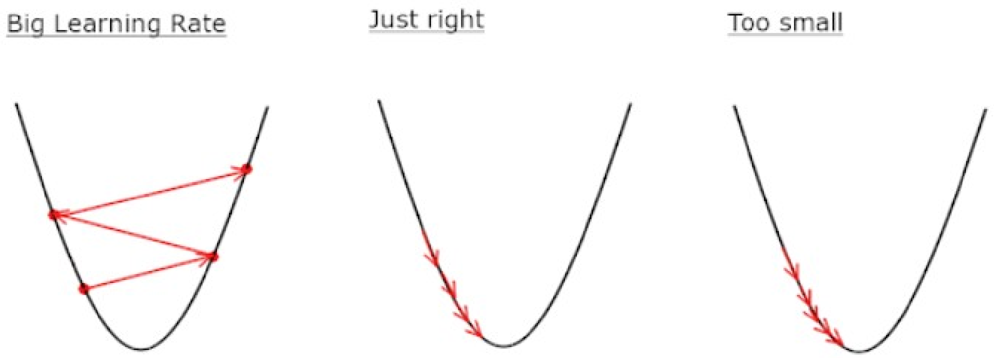
- SGD with momentum is better than only SGD.



$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W, X, y)$$
$$W = W - V_t$$

Importance of learning rate





Fancy optimizer: AdaGrad (Adaptive Gradients)

- Learning rate decay
  - Decay the learning rate for parameters in proportion to their update history (more updates means more decay).

- When h is very big, the parameter will start receiving very small updates.

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$
$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

Fancy optimizer: RMSProp

- Everything is very similar to AdaGrad, except now we decay the denominator as well.

$$h \leftarrow h + \frac{\partial L}{\partial W} \odot \frac{\partial L}{\partial W}$$
$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

$$\Longrightarrow$$

$$h_i \leftarrow \rho h_{i-1} + (1-\rho) \frac{\partial L_i}{\partial W} \odot \frac{\partial L_i}{\partial W}$$
$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

Fancy optimizer: Adam (Adaptive moment)

- RMSProp(learning rate decay) + Momentum

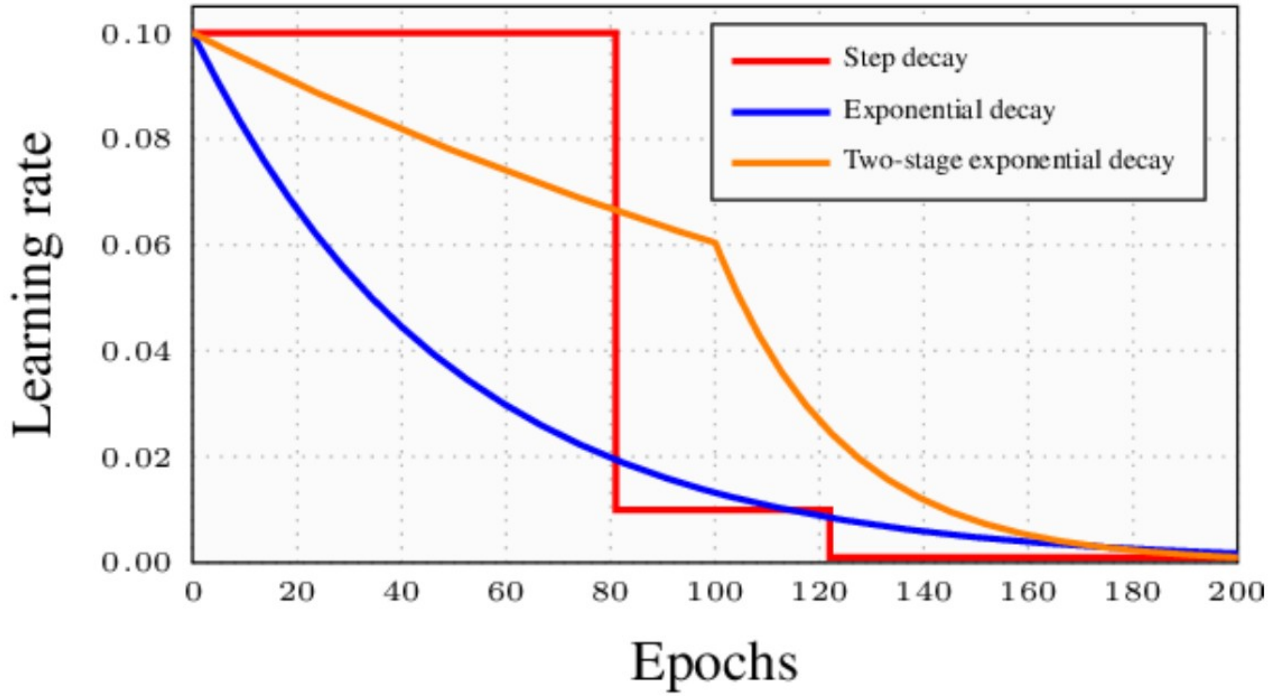
알고리즘 5-5 Adam

입력: 훈련집합 $\mathbb{X}$, $\mathbb{Y}$, 학습률 ρ, 모멘텀 계수 α_1, 가중 이동 평균 계수 α_2
출력: 최적의 매개변수 $\hat{\mathbf{\Theta}}$

난수를 생성하여 초기해 $\mathbf{\Theta}$를 설정한다.
$\mathbf{v} = \mathbf{0}, \mathbf{r} = \mathbf{0}$
t=1
repeat
    그레이디언트 $\mathbf{g} = \frac{\partial J}{\partial \mathbf{\Theta}}\Big|_{\mathbf{\Theta}}$ 를 구한다.
    $\mathbf{v} = \alpha_1 \mathbf{v} - (1 - \alpha_1)\mathbf{g}$ // 속도 벡터
    $\mathbf{v} = \frac{1}{1-(\alpha_1)^t}\mathbf{v}$
    $\mathbf{r} = \alpha_2 \mathbf{r} + (1 - \alpha_2)\mathbf{g} \odot \mathbf{g}$ // 그레이디언트 누적 벡터
    $\mathbf{r} = \frac{1}{1-(\alpha_2)^t}\mathbf{r}$
    $\Delta\mathbf{\Theta} = -\frac{\rho}{\epsilon + \sqrt{\mathbf{r}}}\mathbf{v}$
    $\mathbf{\Theta} = \mathbf{\Theta} + \Delta\mathbf{\Theta}$
    t++
until (멈춤 조건)
$\hat{\mathbf{\Theta}} = \mathbf{\Theta}$

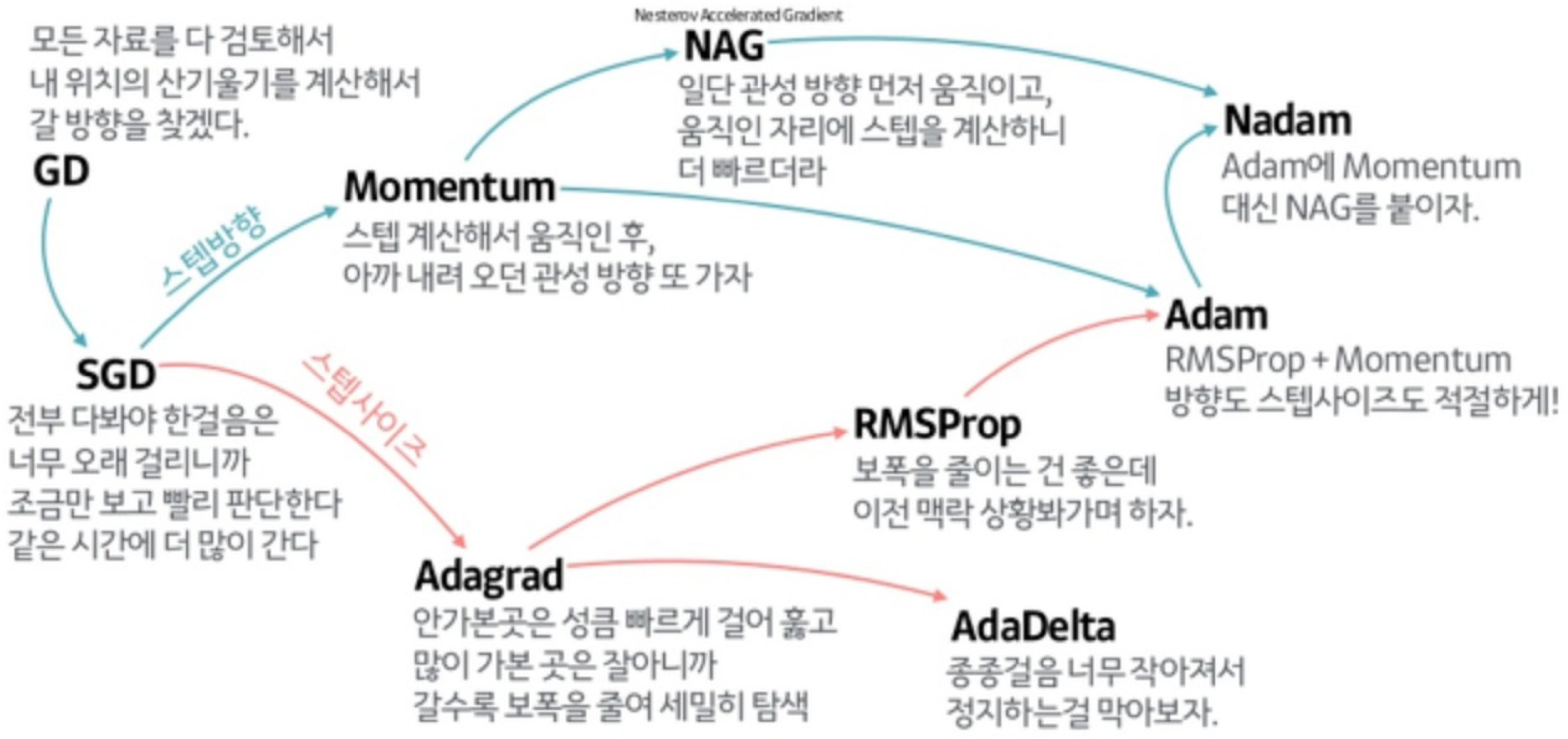
Scheduling learning rate

- SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter.



When we train NN, we typically start with large learning rate and decay over time.

Fancy optimizers



Practice: Deep Learning Techniques

- 다음 기능들을 tensorflow 로 구현해보자 :
  - Weight decay
  - Activation function
  - Batch normalization
  - Weight initialization
  - Optimizers

Thank you :)