Machine & Deep learning basics [AICS305]
Machine learning for cyber security II

고려대학교

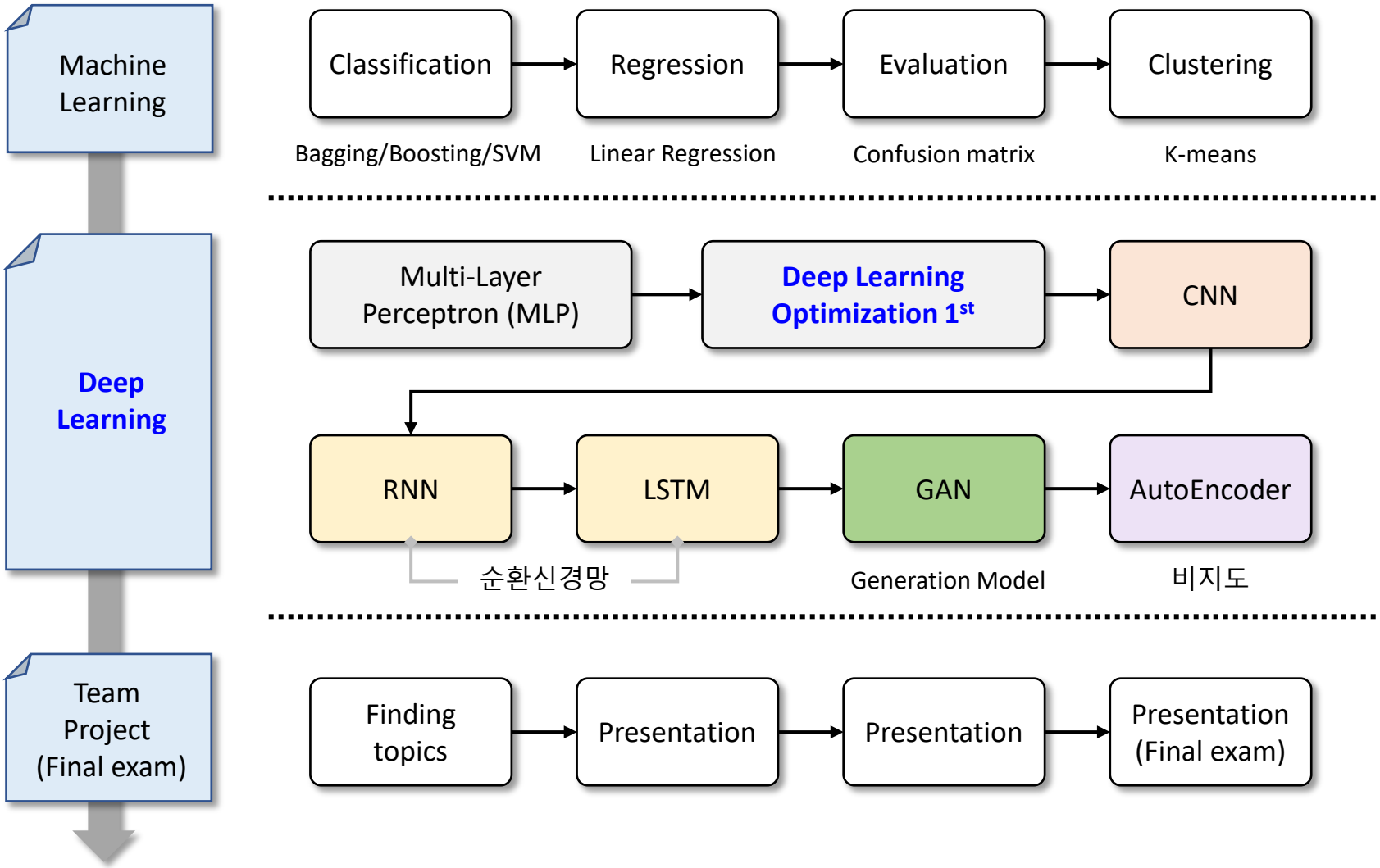
인공지능사이버보안학과

Prof. Mee Lan Han (aeternus1203@gmail.com)

KOREA UNIVERSITY

Machine Learning vs. Deep Learning

■Study Plan



CONTENTS

Loss function

■ Loss function

□ 개념

- 지도학습 시 알고리즘이 예측한 값 (출력값)과 실제 정답의 차이를 비교하기 위한 함수
- 학습 중에 알고리즘이 얼마나 잘못 예측하는지 그 정도를 확인하기 위한 함수

Loss Function = Cost function = Objective Function = Energy Function

□ 특징

- 출력값과 실제 정답이 일치할수록 **손실함수의 값은 작아**지고, 불일치할수록 **손실함수의 값은 커짐**
- 손실 함수는 성능 척도 (Performance Measure)와는 다른 개념

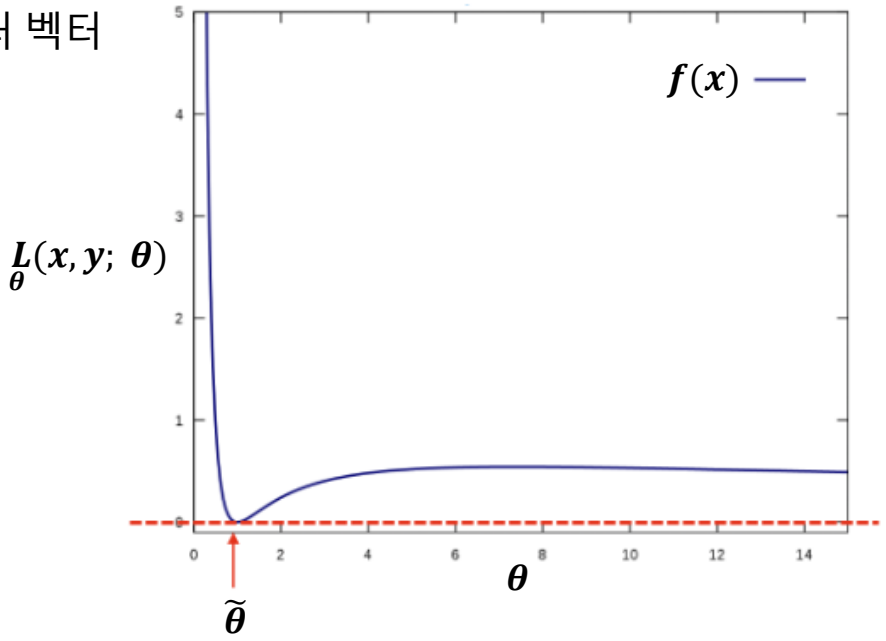
Loss function

■ Loss function 의미

□ Loss function의 수식

- $\boldsymbol{L}$: 손실 함수
- $\boldsymbol{argmin}$: arguments of minimum, 목적 함수를 최소화하는 입력값을 찾는 역할
- $\boldsymbol{x}$: 학습 데이터의 입력값, $\boldsymbol{x}$ 로 얻어낸 예측값 ($\hat{\boldsymbol{y}}$)은 정답 ($\boldsymbol{y}$)과 비교하게 됨
- $\boldsymbol{y}$: 학습 데이터의 정답
- $\boldsymbol{\theta}$: 알고리즘 학습 시 사용되는 모든 파라미터 벡터
- $\tilde{\boldsymbol{\theta}}$: 추정된 최적의 파라미터

$$\tilde{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\boldsymbol{argmin}}\, \boldsymbol{L}(\boldsymbol{x}, \boldsymbol{y};\ \boldsymbol{\theta})$$



✓ 최적화: 매개변수 (가중치, Bias 등)를 조절하여 손실함수 값을 최저로 만드는 과정
✓ 대표적인 최적화 방법 -> **경사하강법**

Loss function

■ Loss function

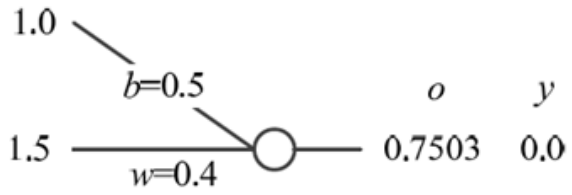
□ 평균 제곱 오차 (Mean Square Error, MSE)

- 오차가 클수록 아래 e 값이 커짐 -> 좋지 않은 모델

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

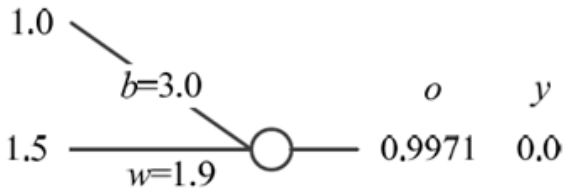
- **MSE가 목적함수로 부적절한 상황** (예시)

(좌) e=0.2815



$$= \frac{1}{2}(0.0 - 0.7503)^2 = 0.281475045$$

(우) e=0.4971



$$= \frac{1}{2}(0.0 - 0.9971)^2 = \mathbf{0.497104205}$$

오차 값이 더 크므로 좋지 않은 모델

Loss function

■ Loss function

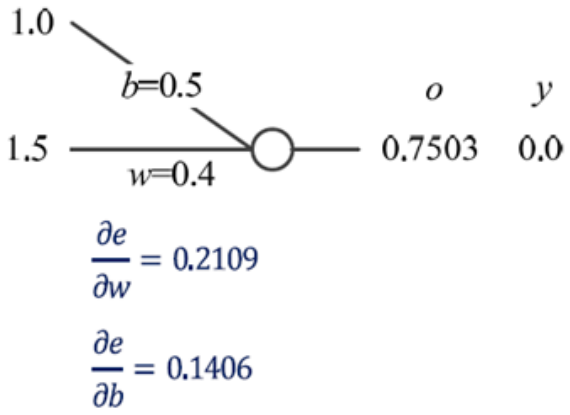
□ 경사하강법 (Gradient Descent)

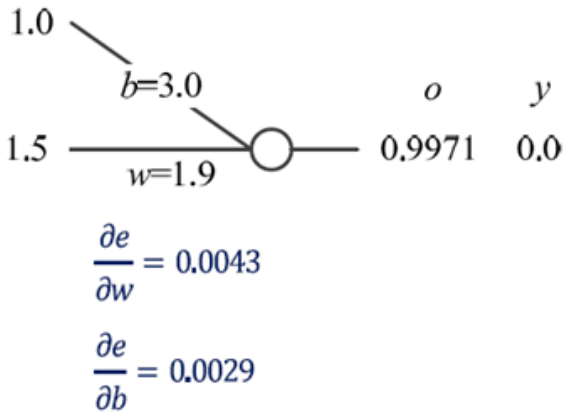
- 손실함수 값을 최저로 만드는 대표적인 최적화 방법

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

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

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





✓ Gradient를 계산해보면 왼쪽의 Gradient가 더 큼

✓ 더 많은 오류를 범한 상황이지만 MSE로 측정할 경우, 더 낮은 벌점을 받게 되는 상황

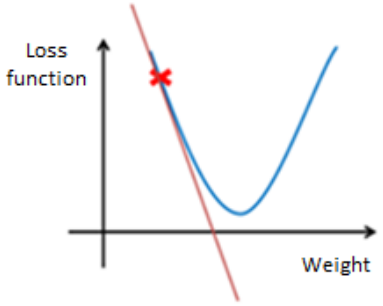
Loss function

■ Loss function

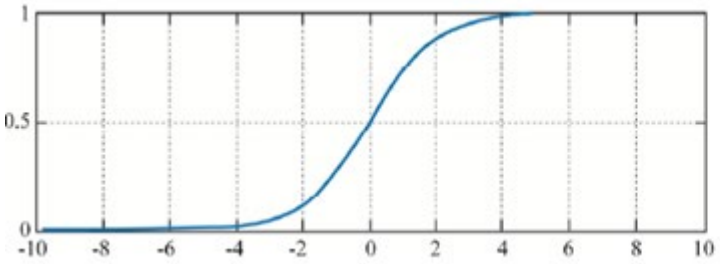
□ 경사하강법 (Gradient Descent)

- 손실함수 값을 최저로 만드는 대표적인 최적화 방법

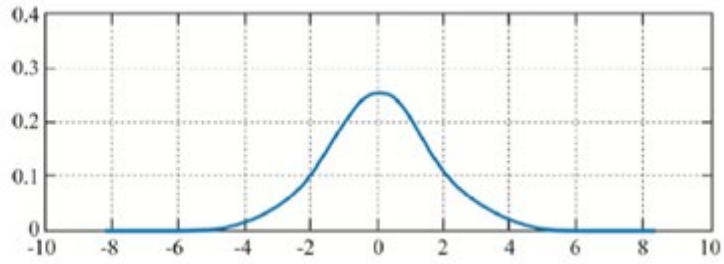
$$e = \frac{1}{2}\|\mathbf{y} - \mathbf{o}\|^2 = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{\sigma}(\boldsymbol{wx} + \boldsymbol{b}))^2$$



- $\boldsymbol{wx} + \boldsymbol{b}$
  - 아래 그래프의 가로축의 값이 커지면 Gradient 가 작아짐



Sigmoid function



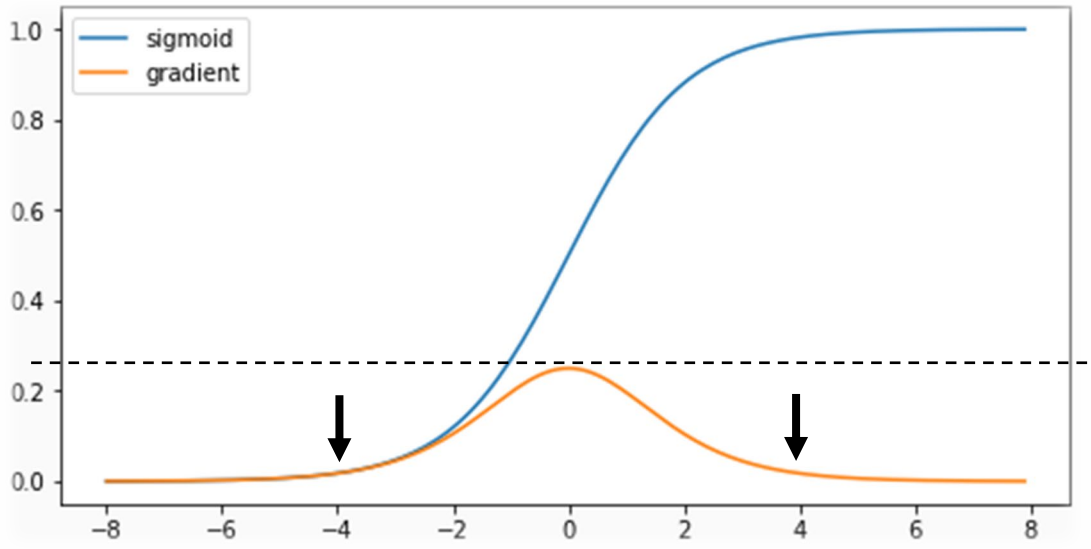
도함수

Loss function

■ Loss function

□ 시그모이드 함수의 미분 최대치는 0.3, 즉, 1보다 작은 값

□ 은닉층이 많아질수록 기울기는 점점 0에 가까워지게 됨



시그모이드 함수를 미분
한 값 최대치 0.3

□ 활성화 (activation)들이 [−4,4] 범위에 들어가지 않게 되면 전체 곱의 기울기는 소멸

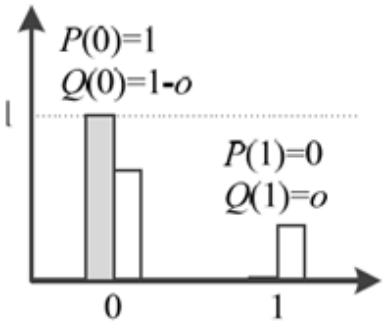
□ 시그모이드 함수를 대체할 수 있는 활성화 함수를 사용하려면 시그모이드 함수와 모양이 비슷하고 경사가 소실되지 않는 함수를 사용 -> ReLU 함수사용

Loss function

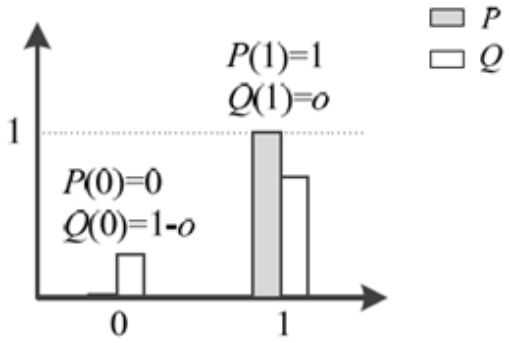
■ Loss function

□ 교차 엔트로피 (Cross Entropy)

- 레이블에 해당하는 y가 확률변수 (**Class가 2**개라고 가정하면 $y \in \{0,1\}$) / **Binary-class classification**
- P는 정답 레이블의 확률 분포2, Q는 신경망 출력의 확률 분포
- Y가 0과 1일 때의 P와 Q의 확률 분포



Y가 0일 때



Y가 1일 때

- $P(0) = 1 - y \qquad Q(0) = 1 - o$
- $P(1) = y \qquad Q(1) = o$

- 교차 엔트로피 수식 $$H(P,Q) = -\sum_{y \in \{0,1\}} P(y) log_2 Q(y)$$

Loss function

■ Loss function

□ 교차 엔트로피 목적함수 (Cross Entropy Loss Function)

- 교차 엔트로피 수식

$$H(P,Q) = -\sum_{y\in\{0,1\}} P(y)log_2 Q(y)$$

- 교차 엔트로피 목적함수

$$e = -(ylog_2 o + (1-y)log_2(1-o)), \quad \text{이때,} \quad e = \sigma(Z)\text{이고,}\ Z = wx + b$$

- y가 1, o가 0.98일 때 (예측이 잘된 경우)

  - **오류** $e = -(1\ \log_2 0.98 + (1-1)\log_2(1-0.98)) = 0.0291$로서 낮은 값 (**오차가 작은 값**)

- y가 1, o가 0.0001일 때 (예측이 엉터리인 경우)

  - **오류** $e = -(1\ \log_2 0.0001 + (1-1)\log_2(1-0.0001)) = 13.2877$로서 높은 값 (**오차가 큰 값**)

Loss function

■ Loss function

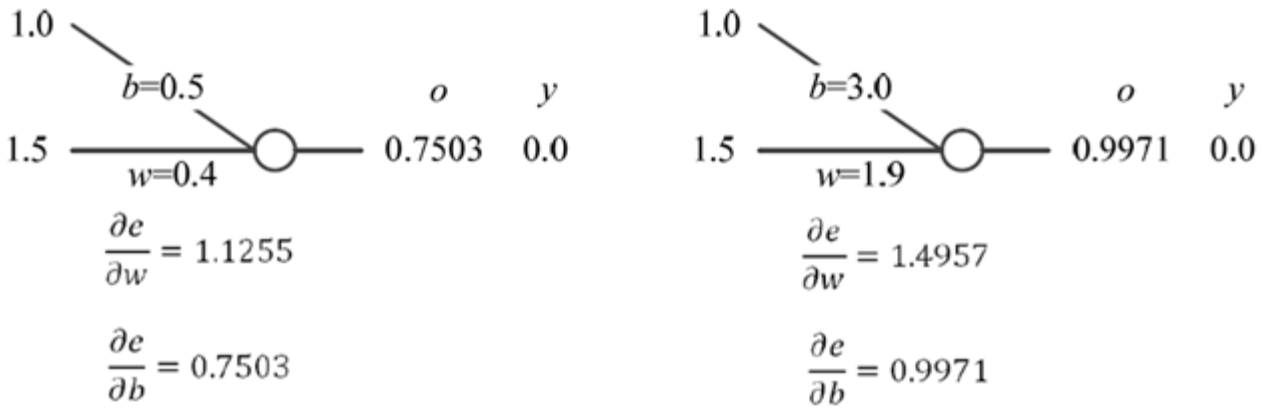
□ 교차 엔트로피 목적함수 (Cross Entropy Loss Function)

- 도함수 측정

$$\begin{aligned}\frac{\partial e}{\partial w} &= -\left(\frac{y}{o}-\frac{1-y}{1-o}\right)\frac{\partial o}{\partial w}\\ &= -\left(\frac{y}{o}-\frac{1-y}{1-o}\right)x\sigma'(z)\\ &= -x\left(\frac{y}{o}-\frac{1-y}{1-o}\right)o(1-o)\\ &= x(o-y)\end{aligned}$$

$$\longrightarrow \quad \left.\begin{aligned}\frac{\partial e}{\partial w} &= x(o-y)\\ \frac{\partial e}{\partial b} &= (o-y)\end{aligned}\right\}$$

- 그레이디언트를 계산해 보면, 오류가 더 큰 오른쪽에 더 큰 벌점 (그레이디언트) 부과



교차 엔트로피를 목적함수로 사용하여 느린 학습 문제를 해결

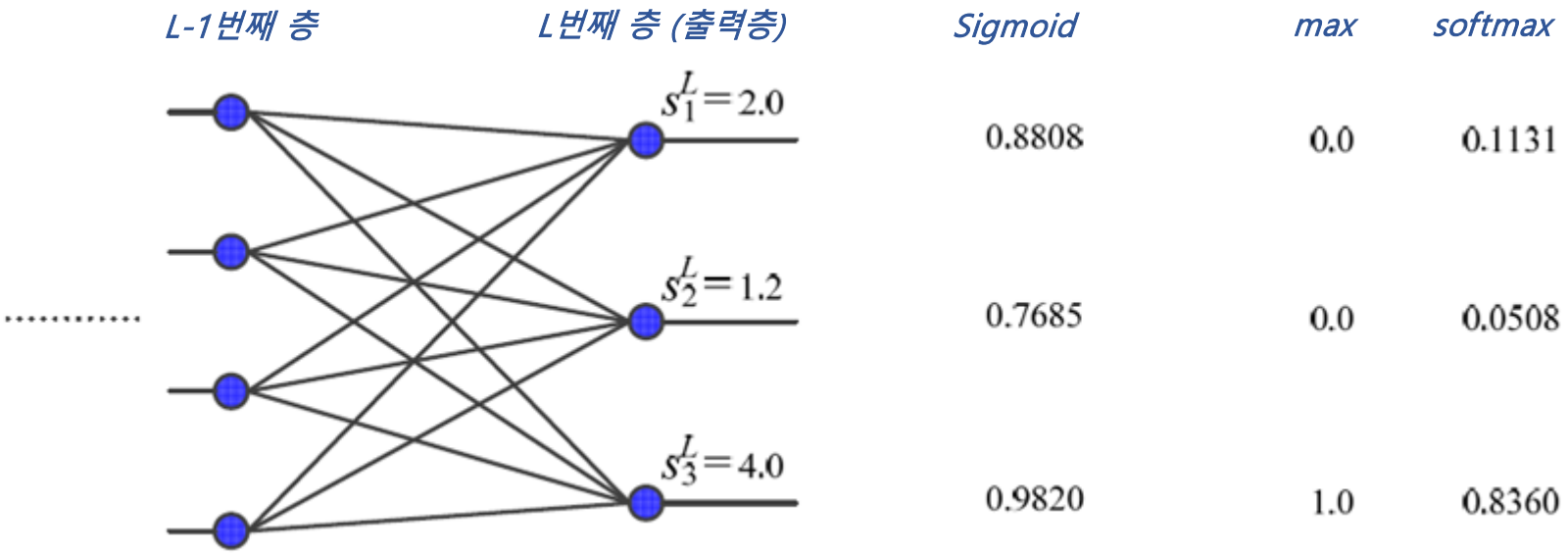
Loss function

■ Loss function

□ Softmax

- max를 모방(출력 노드의 중간 계산 결과 s_i^L 에서 최댓값은 더욱 활성화하고 작은 값은 억제
- 모두 더하면 1이 되어 확률 모방

$$o_j = \frac{e^{S_j}}{\sum_{i=1,c} e^{S_j}}$$



Loss function

■ Loss function

□ 로그우도 목적함수 (Log Likelihood)

- 출력층의 활성화함수로 softmax가 사용됐을 경우, 목적함수로 사용 **/ Multi-class classification**
- 모든 노드를 고려하는 평균 제곱 오차나 교차 엔트로피와는 달리, 정답에 해당하는 노드만 확인

$$e = -log_2 o_y$$

□ Softmax와 로그우도

- Softmax는 최댓값이 아닌 값을 억제하여 0에 가깝게 만든다는 의도 내포
- 학습 샘플이 알려주는 부류에 해당하는 노드만 보겠다는 로그우도와 잘 어울림
- 따라서 둘을 결합하여 사용하는 경우가 많음

```
import numpy as np

y_pred = np.array([0.001, 0.9, 0.001, 0.098])
y_real = np.array([0, 0, 0, 1])

print(-np.log(0.098))
```

CONTENTS

- **Performance improvement**
  - Pre-processing
  - Weight Initialization
  - Gradient Descent & Momentum
  - Adaptive Methods
  - Epoch
  - Batch Normalization
  - Activation Function
  - Stochastic Pooling

Performance improvement

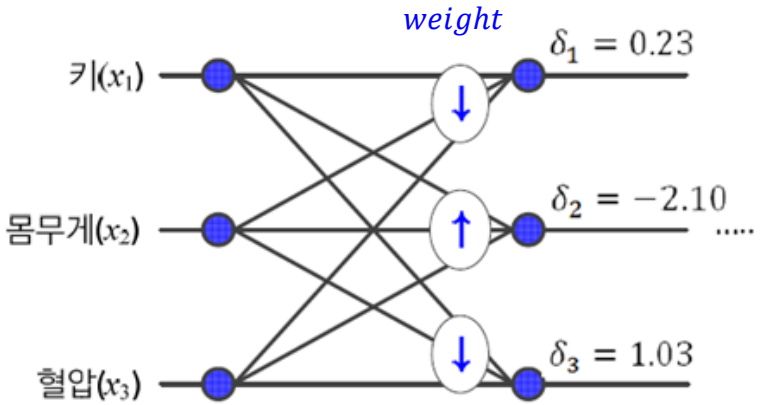
■ 딥러닝의 최적화 기법

(1) 데이터 전처리

□ 데이터 셋 내 각 Feature의 규모 문제

- 예) 건강에 관련된 데이터 (키(m),몸무게(kg),혈압)
  - 키: 1.885m와 1.525m는 **차이는 33cm**/ 특징값을 미터로 기준값 설정했을 때, **차이는 0.33**
  - 몸무게: 65.5kg과 45.0kg은 **20.5라는 차이**
  - 키와 몸무게 Feature 는 대략 62배의 규모 차이

- $-\delta_j z_i$가 gradient 이기 때문에 첫 번째 특징에 연결된 가중치는 두 번째 특징에 연결된 가중치에 비해 대략 62 배 느리게 학습됨



Performance improvement

■ 딥러닝의 최적화 기법

(1) 데이터 전처리

□ 정규화 (Normalization)

- 데이터 값들을 공통된 규정/규범의 간격에 맞게 변경
- Min-Max 정규화
- 데이터들을 0~1 사이의 공통 간격으로 재배치, 최대값=1, 최소값=0으로 설정

$$\mathrm{MinMax} = \frac{\mathrm{data} - \mathrm{data.min}}{\mathrm{data.max} - \mathrm{data.min}}$$

□ 표준화 (Standardization)

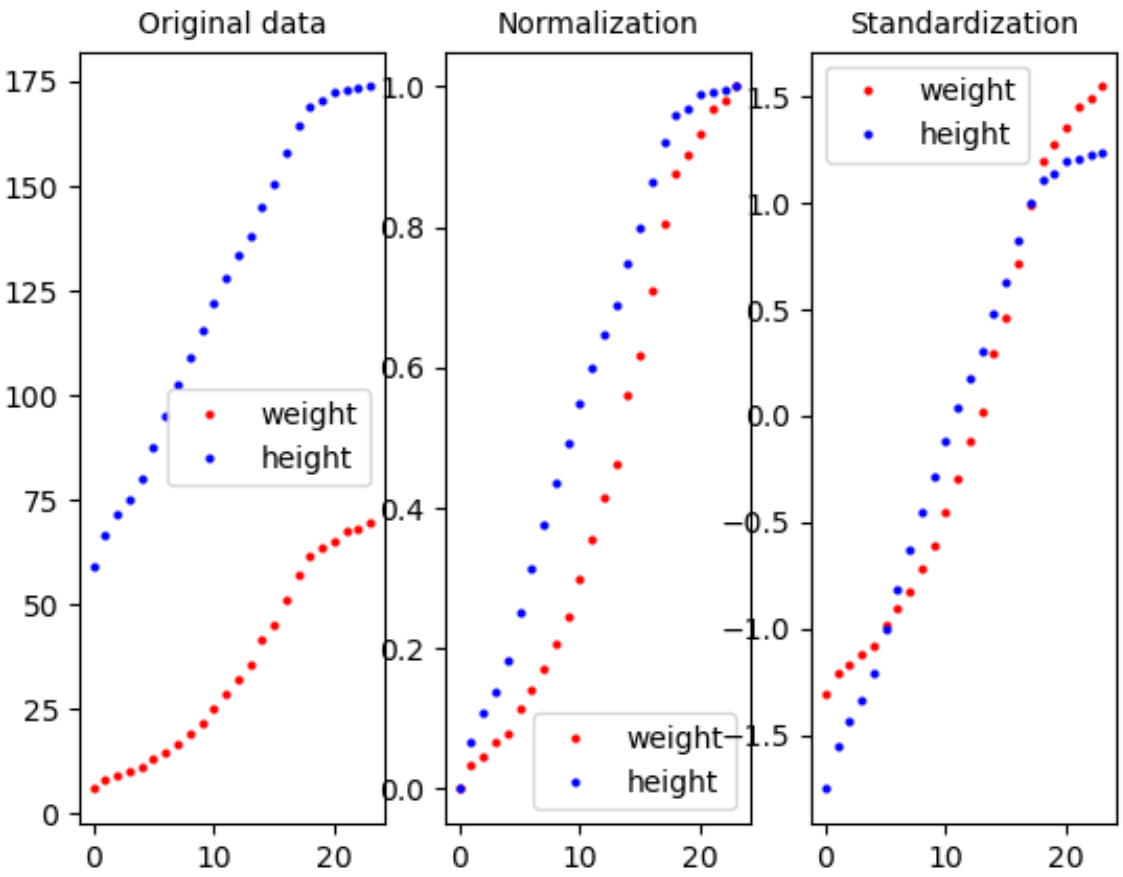
- 데이터 값들을 공통된 근거나 기준인 척도를 변경
- Z-score = 데이터를 데이터의 평균을 뺀 값을 표준편차로 나눔
- 데이터 간격의 크기는 달라질 수 있어도, 데이터 간격의 의미는 달라지지 않음

$$\mathrm{Zscore} = \frac{\mathrm{data} - \mathrm{data.mean}}{\mathrm{data.std}}$$

Performance improvement

■ 딥러닝의 최적화 기법

(1) 데이터 전처리



Performance improvement

■ 딥러닝의 최적화 기법

(1) 데이터 전처리

□ 명칭값 (Norminal value) one-hot 코드로 변환

- Norminal value 은 거리 개념이 없음
- one-hot 코드는 값의 개수만큼 비트를 부여

  - Ex) 성별의 남(1)과 여(2), 체질의 태양인(1), 태음인(2), 소양인(3), 소음인(4)
  - 성별은 2비트, 체질은 4비트 부여

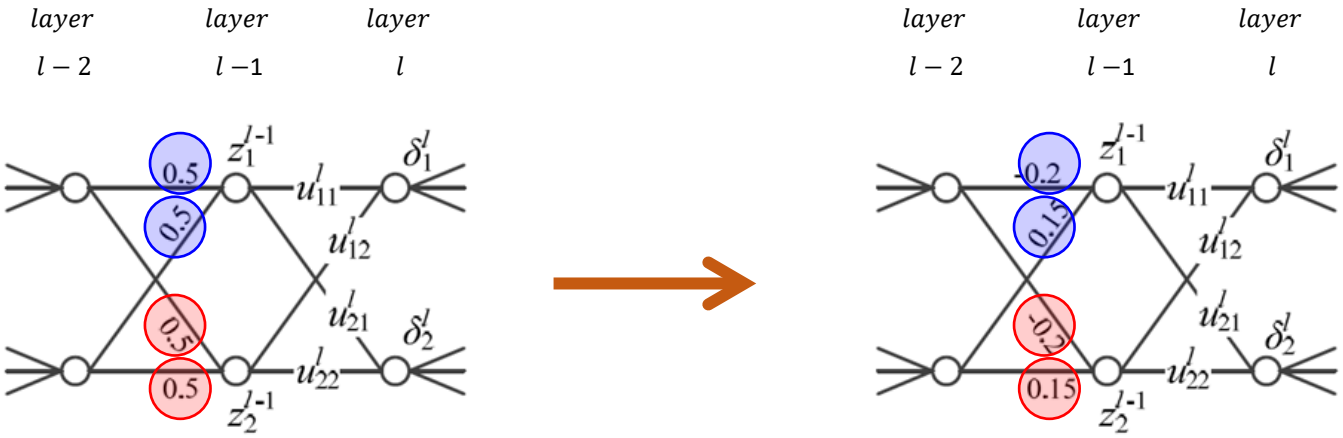
$$(1.755, 65.5, 122, 1, 3) \rightarrow (1.755, 65.5, 122, \underbrace{1, 0}_{\text{성별}}, \underbrace{0, 0, 1, 0}_{\text{체질}})$$

Performance improvement

■ 딥러닝의 최적화 기법

(2) 가중치 초기화

□ 가중치의 초기값을 무엇으로 설정하느냐가 신경망 학습에 영향을 끼치게 됨
□ 각 뉴런의 가중치를 기반으로 error를 결정하기 때문
□ 정확한 모델을 얻기 위해 가장 작은 error가 도출되도록 해야함



- 대칭적 가중치에서는 z_1^{l-1} 과 z_2^{l-1}가 같은 값이 됨
- $-\delta_j z_i$ 가 gradient 이기 때문에 u_{11}^l과 u_{12}^l이 같은 값으로 갱신됨
- 두 노드가 같은 일을 하는 중복성 발생하게 됨

→ 난수로 초기화함으로써 대칭 파괴

Performance improvement

■ 딥러닝의 최적화 기법

(2) 가중치 초기화 (실습 코드)

```
import numpy as np
import matplotlib.ptplot as plt

def sigmoid(x):
return 1 / (1+np.exp(-x))

x = np.random.randn*1000, 100) # 1000개의 데이터
node_num = 100        # 각 은닉층의 노드 수
hidden_layer_sioze = 5          # 은닉층이 5개
activations = {}# 이곳에 활성화값을 저장

for i in range(hidden_layer_size):
        if i !=0:
                x = activations[i-1]
        ## (1)
        w = np.random.randn(node_num, node_num) * 1

        ## (2) 가우시안 분포 N(0,0.01)
        w = np.random.randn(node_num, node_num) * 0.01

        ## (3) Xavier Initialization
        w = np.random.randn(node_num, node_num) / np.sqrt(node_num)
        a = np.dot(x, w)
  z = sigmoid(a)
  activations[i] = z
```

```
for i, a in activations.items():
        plt.subplot(1, len(activations), i+1)
  plt.title(str(i+1) + '-layer')
  plt.hist(a.flatten(), 30, range(0,1))

plt.show()
```

- ✓ 총 5개의 layer
- ✓ 각 layer의 neuron 수는 100개
- ✓ N(0,1)을 따르는 1,000개의 input data
- ✓ activation function: sigmoid
- ✓ activations 변수에 활성화값이 저장

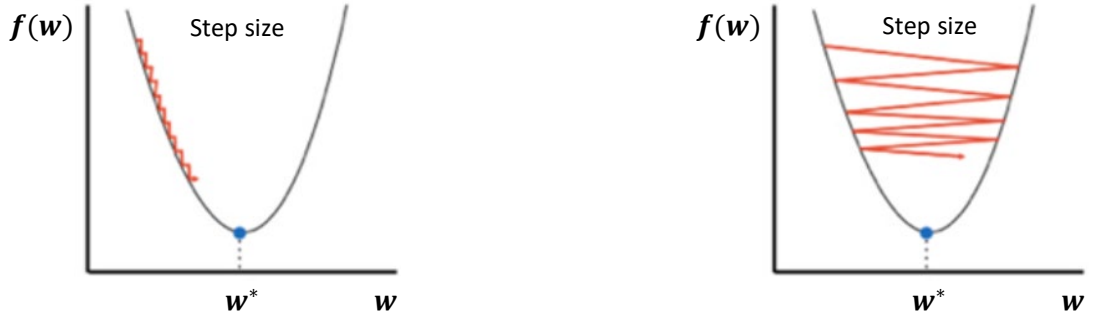
Performance improvement

■ 딥러닝의 최적화 기법

(3) Gradient Descent & Momentum (탄력, 가속도)

□ Gradient Descent

- 미분값 (기울기)이 최소가 되는 지점에 알맞은 가중치 매개변수를 찾아냄
- 비용함수의 경사 반대방향으로 정의된 Step size에 따라 움직이면서 최적의 파라미터를 찾으려 함
- 경사하강법에서는 학습 시 step size 중요



[Reference] https://velog.io/@arittung/DeepLearningStudyDay8

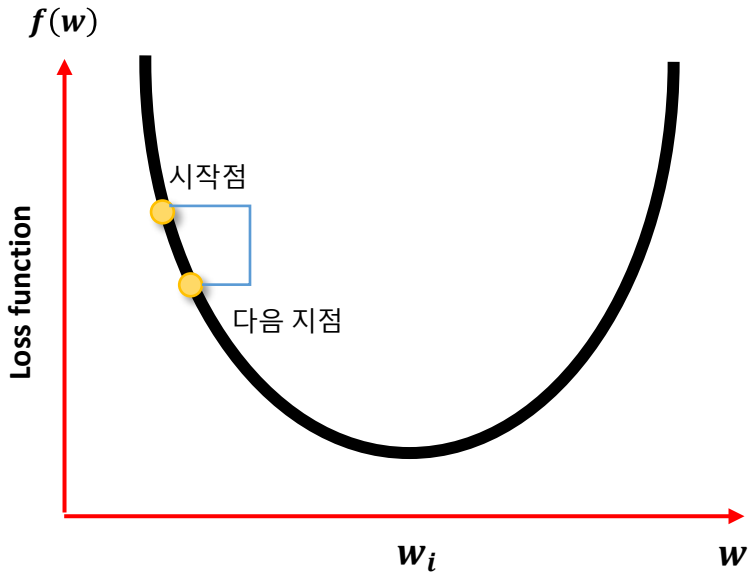
- 학습률 (step size) 이 너무 작을 경우
  - 알고리즘이 수렴하기 위해 반복해야 하는 값이 많으므로 학습시간이 오래 걸림
  - 지역 최소값(local minimum)에 수렴할 수 있음
- 학습률 (step size) 이 너무 클 경우
  - 학습 시간은 적게 걸림
  - step size 너무 커서 전역 최소값(global minimum)을 가로질러 반대편으로 건너뛰어 최소값에서 멀어질 수 있음

Performance improvement

■ 딥러닝의 최적화 기법

(3) Gradient Descent & Momentum (탄력, 가속도)

□ Gradient Descent



$$x_o = x_o - \eta \frac{\partial f}{\partial x_0}$$

η: *Step size (=learning rate)* $\frac{\partial f}{\partial x_0}$: 기울기

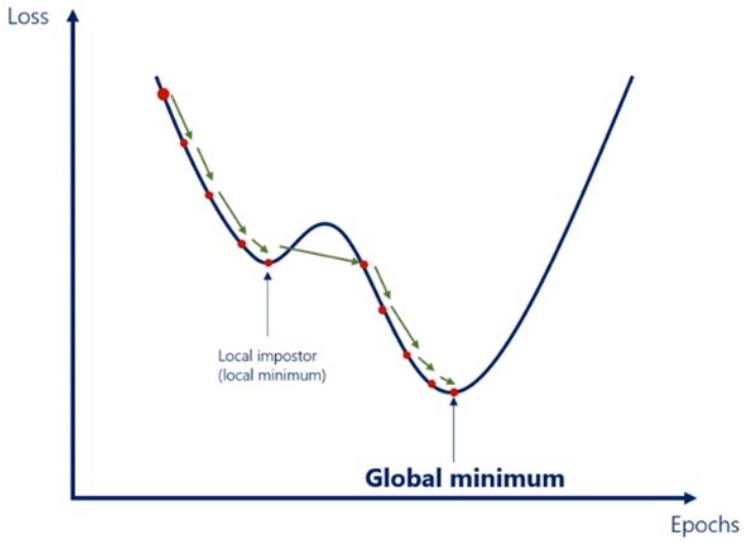
Performance improvement

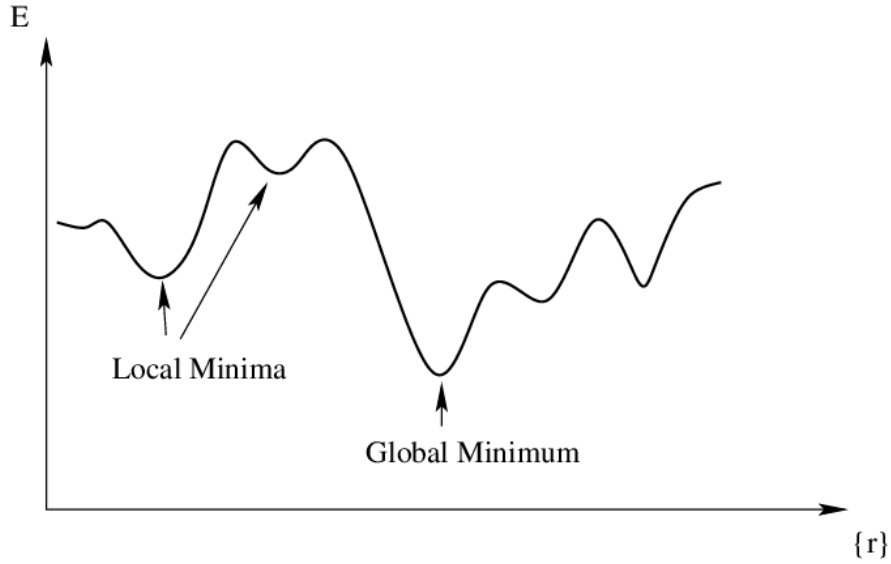
■ 딥러닝의 최적화 기법

(3) Gradient Descent & Momentum (탄력, 가속도)

□ Gradient Descent

- 경사하강법은 현재 위치에서 기울기를 사용하기 때문에 **지역 최소값**에 빠질 수 있음
- **무작위 초기화 (random initialization)로 인해** 알고리즘이 전역 최소값이 아닌 **지역 최소값**에 수렴할 수 있음.





Performance improvement

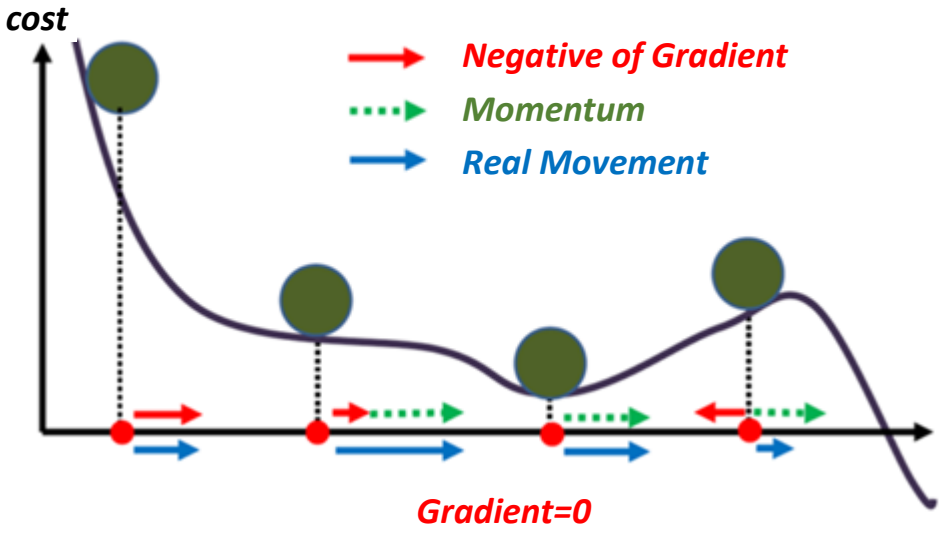
■ 딥러닝의 최적화 기법

(3) Gradient Descent & Momentum (탄력, 가속도)

□ Momentum

- 학습 방향이 바로 바뀌지 않고, 일정한 방향을 유지하려는 성질
- 같은 방향의 학습이 진행된다면 가속을 가지며 더 빠른 학습을 기대할 수 있음

- **Movement = Negative of Gradient + Momentum**
  - 기울기에 관성을 부과하여 작은 기울기는 쉽게 넘어갈 수 있도록 만들어 줌
  - 공의 관성을 이용하여 쉽게 넘어갈 수 있게 하여 지역 최소값을 탈출t
  - 모멘텀을 사용하지 않으면 기울기가 매우 작은 구간을 빠져나오는데 아주 오랜 시간이 걸림



[Reference] https://velog.io/@arittung/DeepLearningStudyDay8

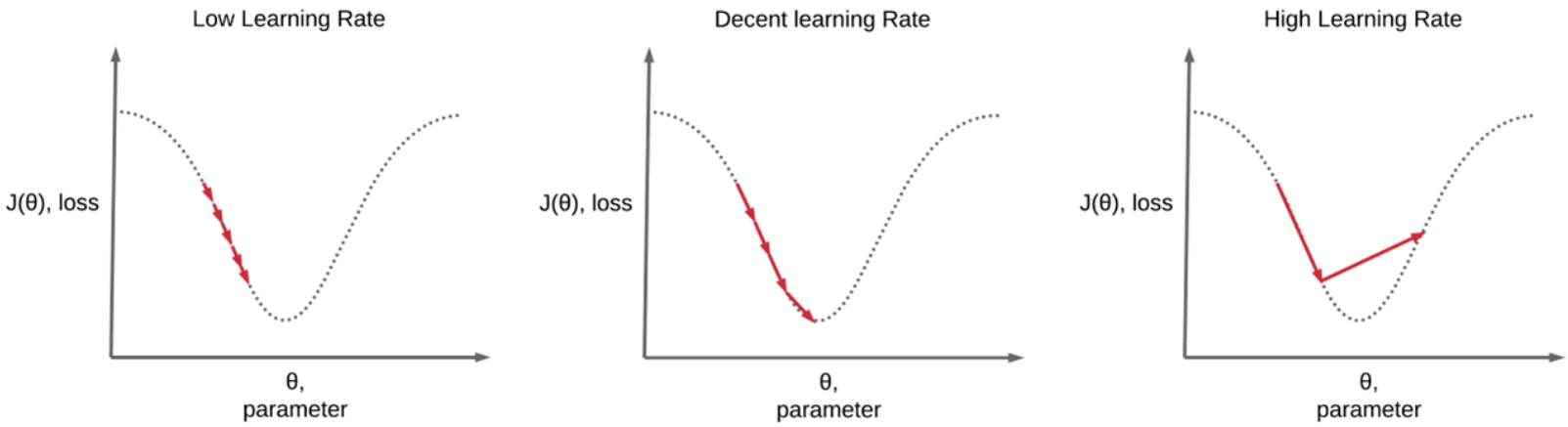
Performance improvement

■ 딥러닝의 최적화 기법

(4) 적응적 학습률 (Adaptive Methods)

□ 학습률 (step size) (Momentum) 의 중요성

- 학습률이 너무 작으면 학습에 많은 시간이 소요
- 학습률이 너무 크면 진동할 가능성이 높음 (오버슈팅에 따른 진자 현상)



□ 적응적 학습률

- Momentum는 질량(m) * 속도(v)
- 신경망에서는 질량을 1로 가정하여 속도만 나타냄
- 적응적 학습률은 매개변수마다 자신의 상황에 따라 학습률을 조절해 사용

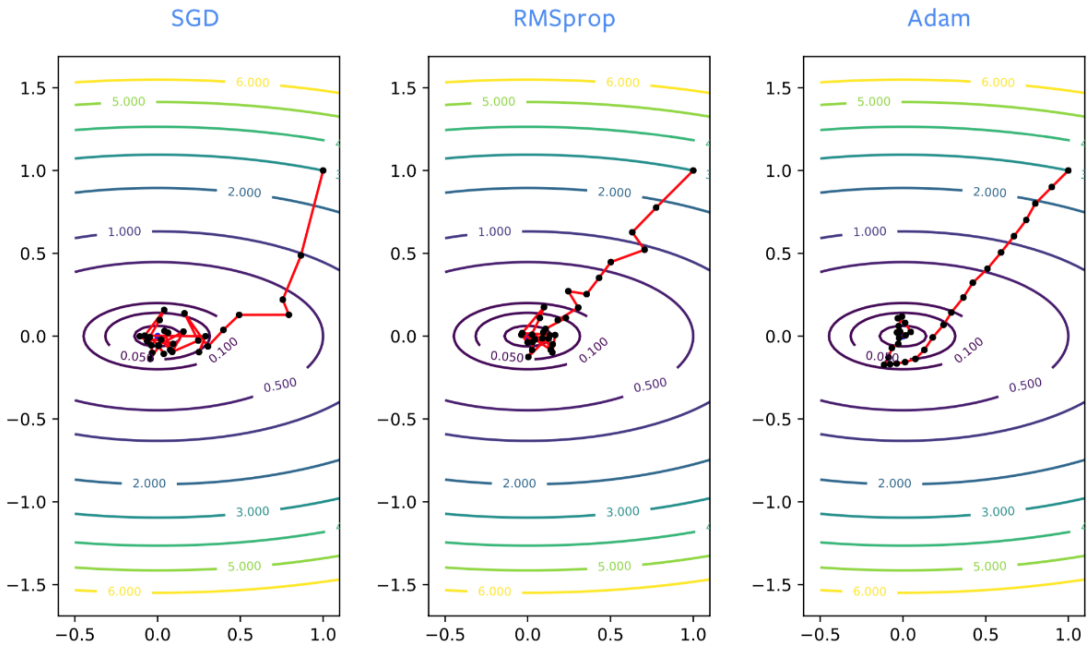
Performance improvement

■ 딥러닝의 최적화 기법

(4) 적응적 학습률 (Adaptive Methods)

□ 적응적 학습률

- **SGD:** 네트워크 상 모든 개별 가중치는 동일한 학습률 사용
- **Adagrad:** 이전 gradient 를 누적한 정보 벡터 r를 이용, 이전 gradient 누적값이 크면, 다음에는 파라미터가 조금만 이동하고, 반대로 누적값이 작으면 많이 이동
- **RMSprop:** 이전 gradient를 누적할 때 오래된 것의 영향을 지수적으로 줄이기 위해 가중 이동평균 기법을 사용하여 Adagrad를 개선한 기법
- **Adam:** RMSProp에 모멘텀을 적용하여 RMSprp를 개선한 기법, 가장 일반적으로 사용되는 기법

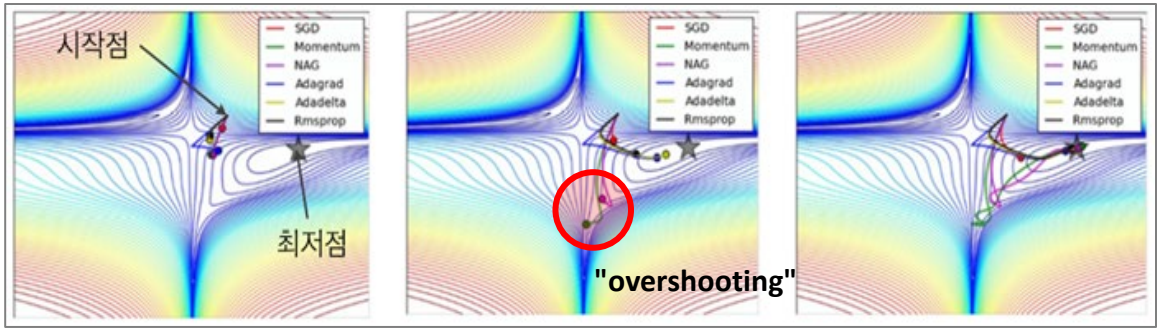


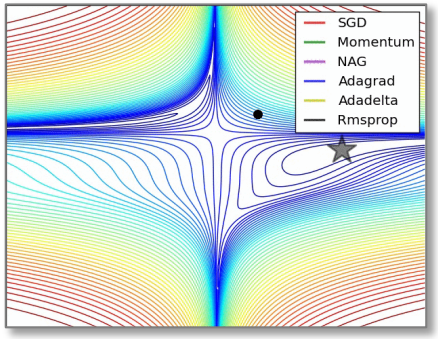
Performance improvement

■ 딥러닝의 최적화 기법

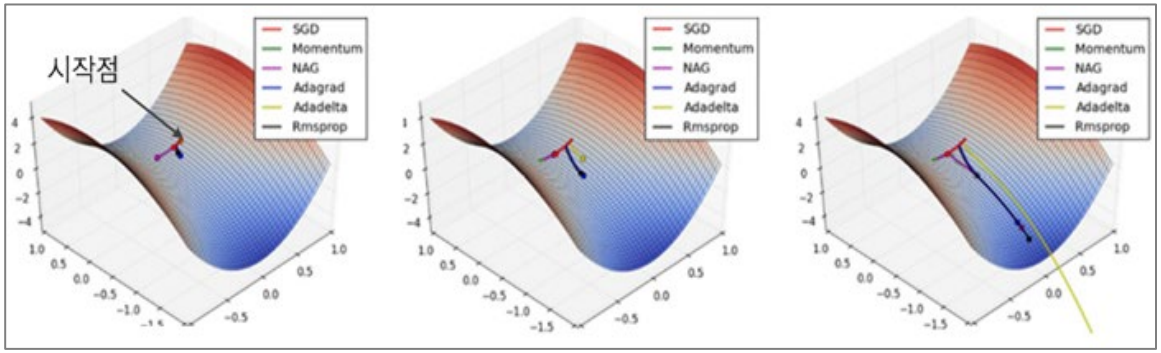
(4) 적응적 학습률 (Adaptive Methods)

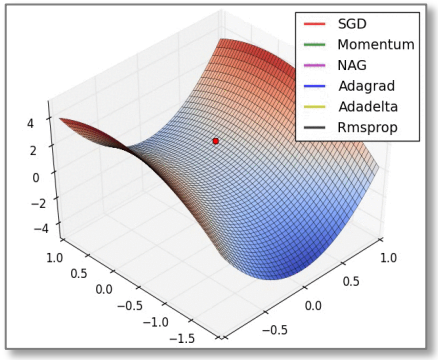
- □ SGD: 대칭을 깨는데 매우 어려움을 겪고 상단에 고정
- □ RMSprop: 안장 방향에서 매우 낮은 기울기, RMSprop 업데이트의 분모 항으로 인해 이 방향을 따라 효과적인 학습 속도가 증가





(a) 중앙으로 급강하하는 절벽 (협곡) 지형





(b) 중앙 부근에 안장점이 saddle point 있는 지형

[reference] https://cs231n.github.io/neural-networks-3/

Performance improvement

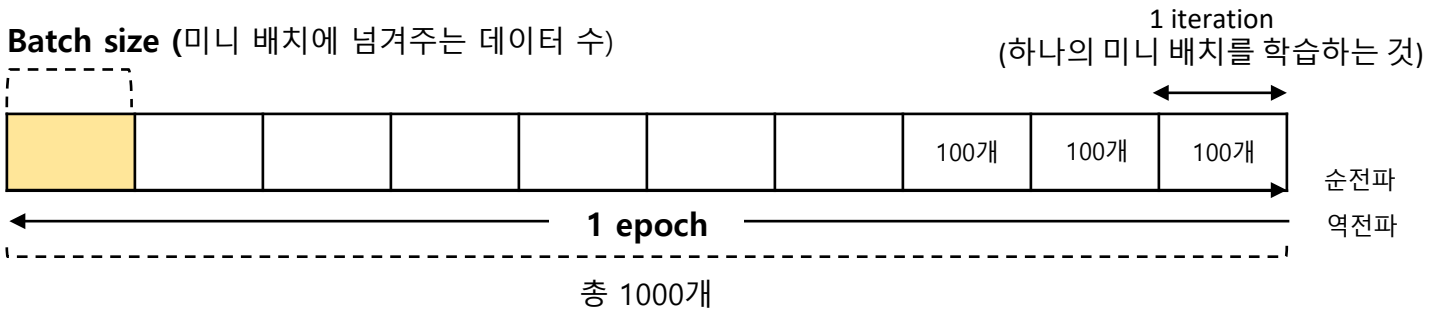
■ 딥러닝의 최적화 기법

(5) Epoch (에포크)

□ 훈련 데이터 셋에 포함된 모든 데이터들이 한 번씩 모델을 통과한 횟수
□ 학습데이터 전체를 한번 학습하는 것

□ 1 epoch
- 전체 학습 데이터 셋이 한 신경망에 적용되어 순전파와 역전파를 통해 신경망을 한 번 통과했다는 의미
- 10 epoch 은 동일한 학습 데이터 셋을 10회 모델에 학습시켰다는 의미

□ epoch 값이 높을수록 다양한 무작위 가중치로 학습하였다는 의미
□ 지나치게 epoch이 올라가면 학습 데이터 셋에 대한 과적합 발생



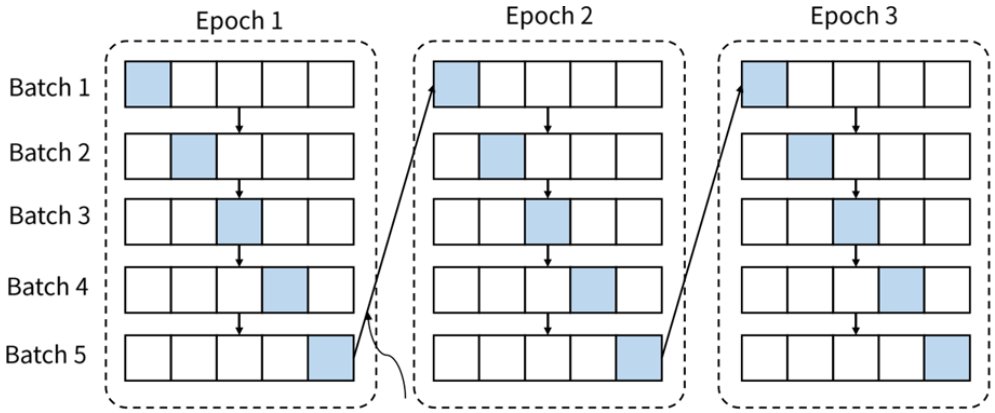
Performance improvement

■ 딥러닝의 최적화 기법

(5) Epoch (에포크)

□ Batch size

- Batch는 단일 반복에서 기울기를 계산하는데 사용하는 data의 총 개수, **Gradient를 구하는 단위**
- 연산 한번에 들어가는 데이터 크기
- mini Batch: 1 Batch size에 해당하는 데이터 셋



- **배치 사이즈가 너무 큰 경우**
  - ✓ 한번에 처리해야 할 데이터의 양이 많음
  - ✓ 학습 속도가 느려지고, 메모리 부족 문제가 발생 가능

- **배치 사이즈가 너무 작은 경우**
  - ✓ 적은 데이터를 대상으로 가중치를 업데이트하고 이 업데이트가 자주 발생하므로 훈련이 불안정

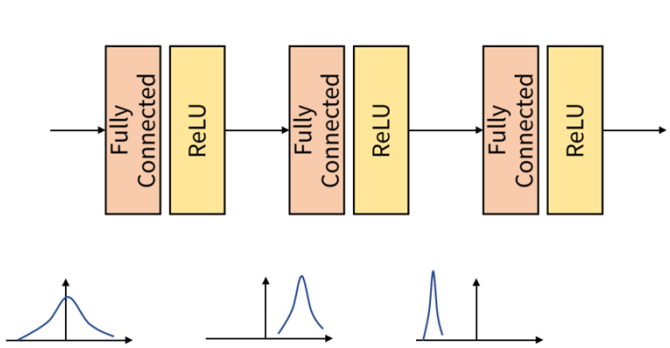
Performance improvement

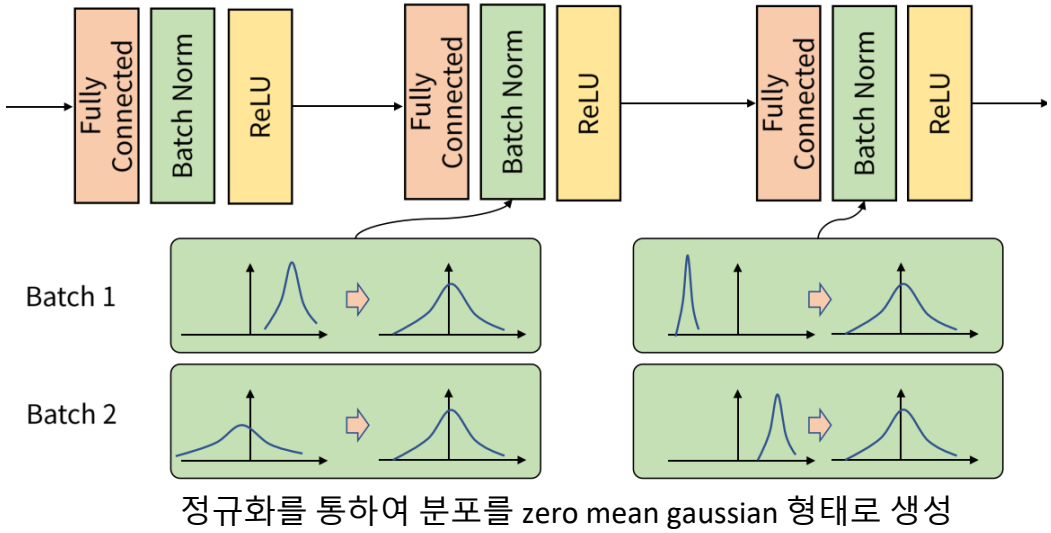
■ 딥러닝의 최적화 기법

(6) Batch Normalization

□ Internal Covariate Shift 현상 (내부 공변량 이동)

- Batch 단위로 학습을 하게 되면 발생하는 문제점
- 학습 과정에서 계층 별로 입력의 데이터 분포가 달라지는 현상
-
- 한 레이어 마다 입력과 출력을 가지고 있는데 이 레이어들끼리 covariate shift가 발생
- 레이어가 깊어질 수록 분포 형태가 더 변화가 심함 → Batch Normalization을 이용





정규화를 통하여 분포를 zero mean gaussian 형태로 생성

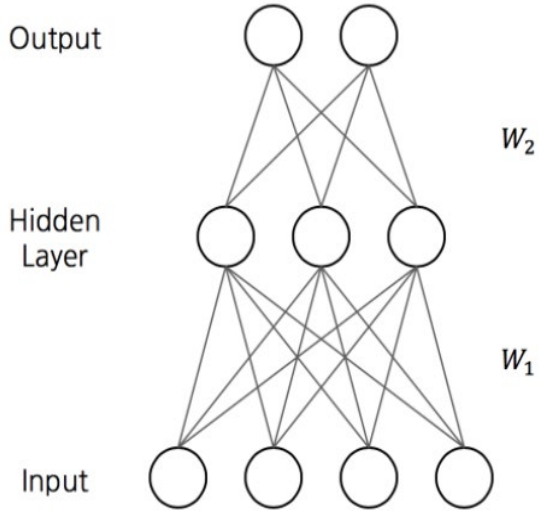
- [Reference] https://gaussian37.github.io/dl-concept-batchnorm/

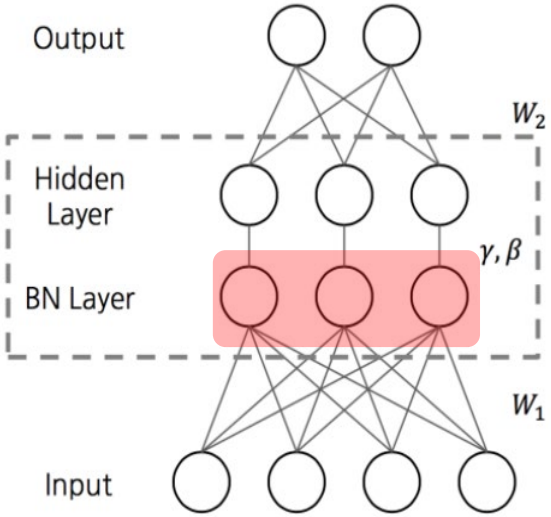
Performance improvement

■ 딥러닝의 최적화 기법

(6) Batch Normalization

- 학습 과정에서 각 배치 단위 별로 데이터가 다양한 분포를 가지더라도 각 배치별로 평균과 분산을 이용해 정규화하는 과정
- Batch마다 입력값의 범위를 스케일링하는 과정
- 장점
  - **데이터 Scale 통일**: 모든 Layer의 Feature가 동일한 Scale이 되도록 함
  - **활성화 함수 맞춤형 분포 변화**: 추가적인 스케일링(γ)과 편향(β)을 학습함으로써 활성화 함수 종류에 맞게 적합한 분포로 변환 가능





$$BN(X) = \gamma\left(\frac{X - \mu_{batch}}{\sigma_{batch}}\right) + \beta$$

Ref) http://sanghyukchun.github.io/88/

Performance improvement

■ 딥러닝의 최적화 기법

(6) Batch Normalization

□ 배치 정규화의 장점

- 학습률을 높게 설정할 수 있어서 학습 속도가 개선됨
- 과적합 (Overfitting) 억제 / 기울기 소실 (Gradient vanishing) 방지

□ 배치 정규화 알고리즘 (수식적인 접근)

Mini-batch 평균

$$\mu_B \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$$

- 미니배치 $B = x_1, x_2, \dots, x_n$ 을 평균이 0, 분산이 1인 표준정규분포를 따르는 $\hat{B} = \widehat{x_1}, \widehat{x_2}, \dots, \widehat{x_n}$로 정규화

Mini-batch 분산

$$\sigma^2{}_B \leftarrow \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$$

- 정규화에서 $x_i - \mu_B$ 가 평균을 0으로 설정

Normalization

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma^2{}_B + \epsilon}}$$

- $\sqrt{\sigma^2{}_B + \epsilon}$ 가 분산을 1로 설정
- 표준편차로 나누면 분산이 1이 됨
- ϵ은 0으로 나뉘어지는 것을 방지하기 위한 상수

Scale and shift

$$y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta(x_i)}$$

- 배치 정규화 계층마다 데이터에 대한 고유한 스케일링(Scaling, γ), 쉬프팅(Shifting, β) 수행

Performance improvement

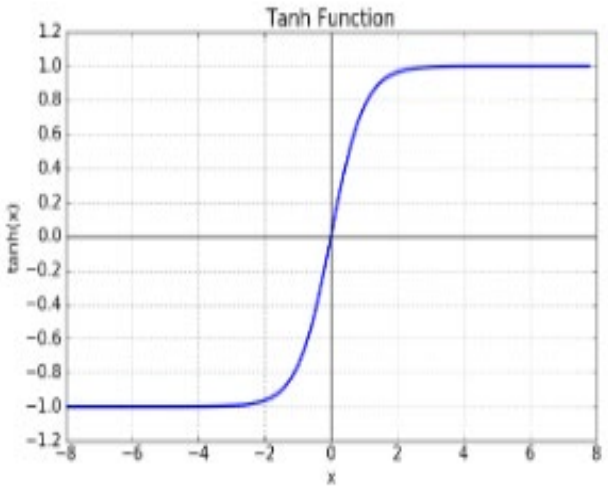
■ 딥러닝의 최적화 기법

(7) ReLU (Rectified Linear Unit) 활성함수

□ tanh(x) 함수 (시그모이드와 비슷) 를 사용하면 미분값의 범위가 확장
□ 고차원 데이터를 다룰 경우에는 값이 커질 수 있어 다시 경사가 소실될 수 있음
□ 복잡한 데이터일수록 고차원일 경우가 많은데 이를 회피할 수 있는 활성화 함수가 ReLU 함수

$$\varphi(\mathbf{x}) = \begin{cases} \mathbf{x}, \mathbf{x} > \mathbf{0} \\ \mathbf{0}, \mathbf{x} \leq \mathbf{0} \end{cases}$$

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



$$f'(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$$

ReLU 함수의 도함수
(미분한 식)

- x (입력값)가 0보다 작을 때 (음수)는 모든 값을 0으로 출력
- x (입력값)가 0보다 클 때 (양수)는 입력값을 그대로 출력
- **x가 0보다 크기만 하면 미분값이 1**이 되기 때문에 여러 은닉층을 거쳐도 맨 처음 층까지 사라지지 않고 남아있음
- 딥러닝의 발전에 기여!!!

Performance improvement

■ 딥러닝의 최적화 기법

(7) ReLU (Rectified Linear Unit) 활성함수

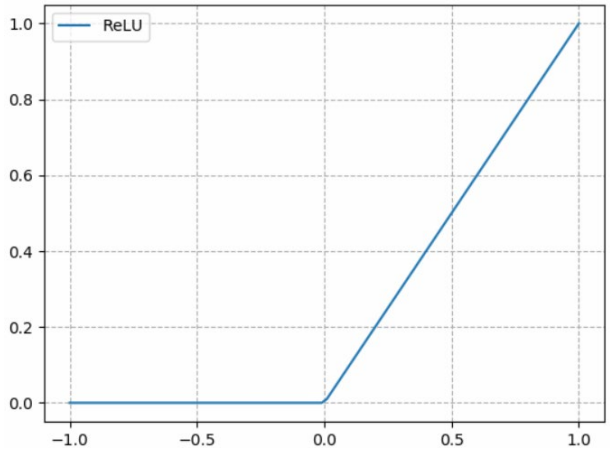
□ tanh(x) 함수 (시그모이드와 비슷) 를 사용하면 미분값의 범위가 확장

□ 고차원 데이터를 다룰 경우에는 값이 커질 수 있어 다시 경사가 소실될 수 있음

□ 복잡한 데이터일수록 고차원일 경우가 많은데 이를 회피할 수 있는 활성화 함수가 ReLU 함수

$$\varphi(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

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



$$f'(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$$

ReLU 함수의 도함수 (미분한 식)

- ReLU 함수의 도함수는 x가 아무리 커져도 1을 반환하므로 경사가 소실되지 않음
- 시그모이드 함수나 쌍곡탄젠트 함수에 비교해 학습 속도가 빠름
- ReLU와 ReLU의 도함수는 단순한 식으로 표현되기 때문에 빠르게 계산이 가능
- 단점으로는 x≦0일 때는 함수값도 경사도 0이기 때문에 ReLU를 활성화 함수로 사용한 신경망 모델의 뉴런 중 활성화되지 못한 뉴런은 학습 동안 활성화가 되지 않는 문제가 있음

Thank you

KOREA
UNIVERSITY

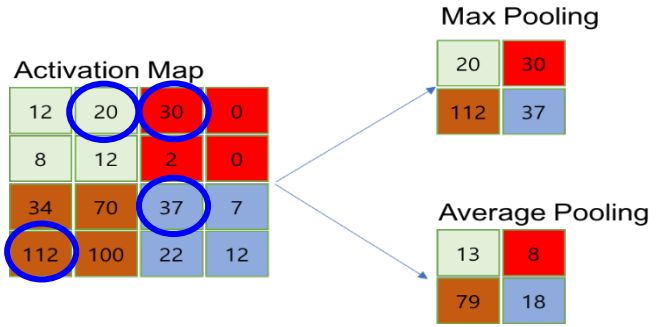
Performance improvement

■ 딥러닝의 최적화 기법

(8) Stochastic Pooling

□ Pooling

- sub-sampling을 이용하여 feature map의 크기를 줄이고, 위치나 이동에 강한 특징을 추출하기 위한 방법
- **Max pooling**: pooling window 내의 가장 큰 값을 선택하는 방법, Overfitting 되는 단점
- **Average pooling**: 평균 연산으로 인해 학습 결과가 좋지 않음



□ **Stochastic pooling**

- 최대값 또는 평균값 대신 확률에 따라 적절한 activation을 선택함
- 확률값은 특정 activation에 대해 전체의 activation의 합을 나누는 방식으로 계산됨

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

- Dropout과 같이 다양한 네트워크를 학습하는 듯한 model average 효과를 얻을 수 있음