Deep Learning 101

2주차

12기 이두형
12기 임효진

KU-BIG

Curriculum

1주차 : 딥러닝 소개 및 기초 (XOR문제, 퍼셉트론, 활성화 함수 등)

2주차 : Multi-layer Neural Network (Loss Function, Gradient Descending, Backpropagation, MNIST practice, Optimization)

3주차 : CNN 소개 및 기초 (Convolution, Padding, Stride, Pooling등 기초 개념 소개)

4주차 : CNN 실습 (세션 후 조별 과제 부여)

5주차 : RNN, LSTM, GRU

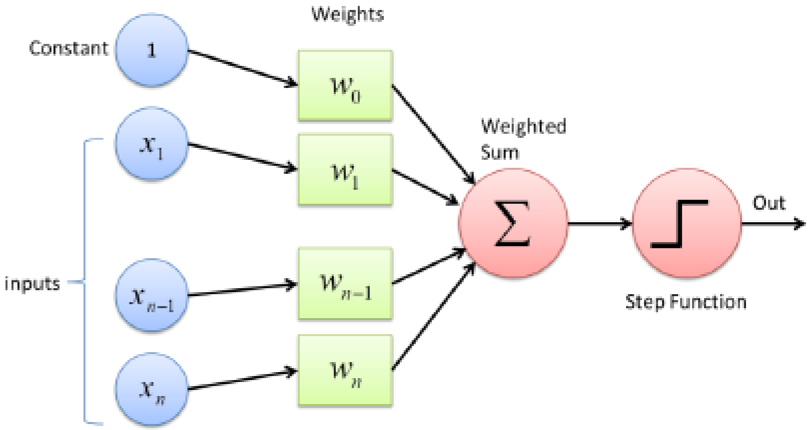
6주차 : seq2seq, 실습 (세션 후 조별 과제 부여)

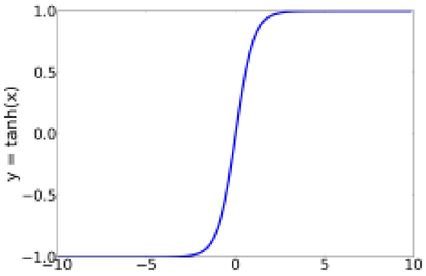
7주차 : 조별 과제 발표

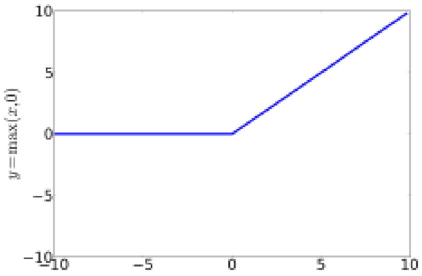
Review

Review

- Perceptron

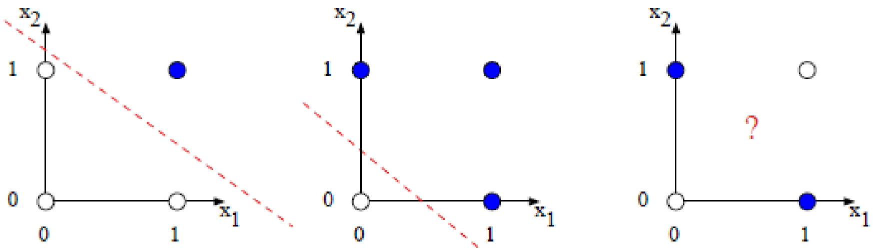


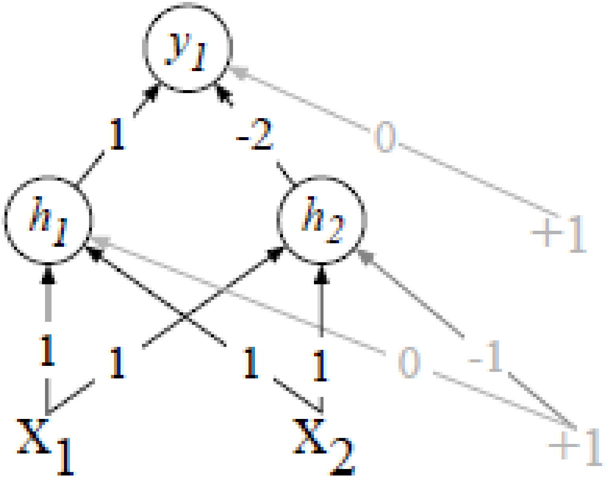




Review

- Multi-Layer Perceptron





Week2

Loss Function

- Maximum Likelihood

$$P(x|\theta) = \prod_{k=1}^{n} P(x_k|\theta)$$

$$L(\theta|x) = \log P(x|\theta) = \sum_{i=1}^{n} \log P(x_i|\theta)$$

Loss Function

- Loss Function

$$
\begin{aligned}
L_{CE}(\hat{y}, y) &= -\sum_{k=1}^{K} \mathbb{1}\{y = k\} \log \hat{y}_i \\
&= -\sum_{k=1}^{K} \mathbb{1}\{y = k\} \log \hat{p}(y = k|x) \\
&= -\sum_{k=1}^{K} \mathbb{1}\{y = k\} \log \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}
\end{aligned}
$$

Loss Function

- Loss Function : 예측 값과 실제 값의 차이를 보여주는 지표

- Learning의 목적 : Loss Function의 최소화 -> 얼마나 잘 예측 하느냐?

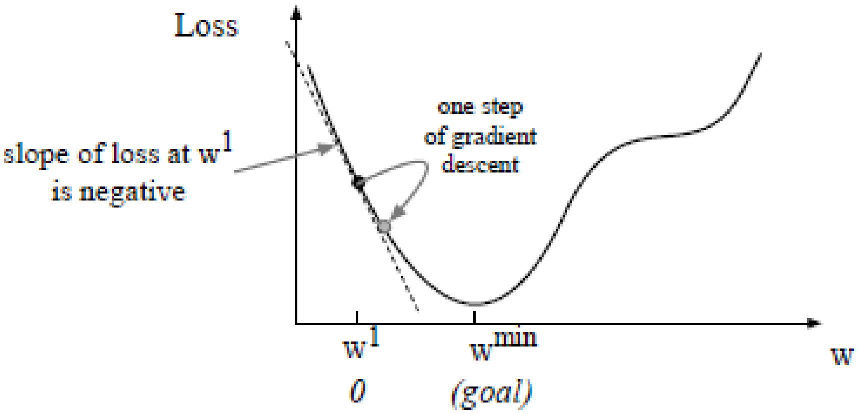
Gradient Descending

- Loss Function이 최소가 되는 점을 찾고 싶다.

- 미분?

$$L_{CE}(\hat{y},y) = -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))]$$

Gradient Descending & Learning Rate



$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x; w)$$

$$\nabla_\theta L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$

Gradient Descending & Learning Rate

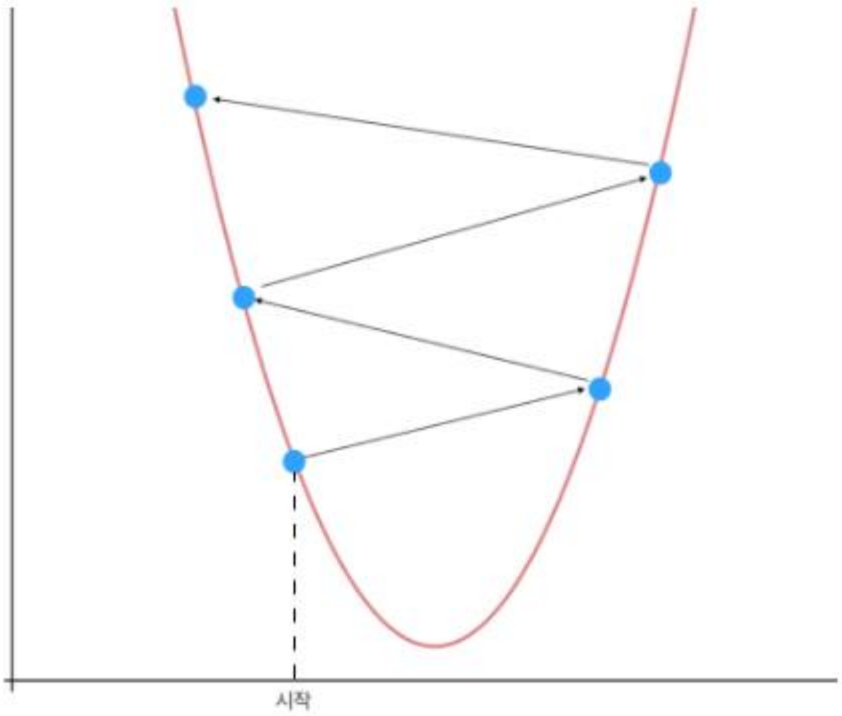
eta_large

```
import numpy as np
def E(w):
  return 3*(w-2)**2+5

def dE(w):
  return 6*(w-2)

eta=0.4
w=np.zeros((11,1))
w[0]=5
print('w(0)=%f' %(w[0]), 'E(w(0))=%f' %(E(w[0])))
for i in range(0,10):
  w[i+1]=w[i] - eta*dE(w[i])
  print('w(%d)=%f' %(i+1,w[i+1]), 'E(w(%d))=%f' %(i+1,E(w[i+1])))
```

```
w(0)=5.000000 E(w(0))=32.000000
w(1)=-2.200000 E(w(1))=57.920000
w(2)=7.880000 E(w(2))=108.723200
w(3)=-6.232000 E(w(3))=208.297472
w(4)=13.524800 E(w(4))=403.463045
w(5)=-14.134720 E(w(5))=785.987568
w(6)=24.588608 E(w(6))=1535.735634
w(7)=-29.624051 E(w(7))=3005.241843
w(8)=46.273672 E(w(8))=5885.474012
w(9)=-59.983140 E(w(9))=11530.729064
w(10)=88.776396 E(w(10))=22595.428965
```

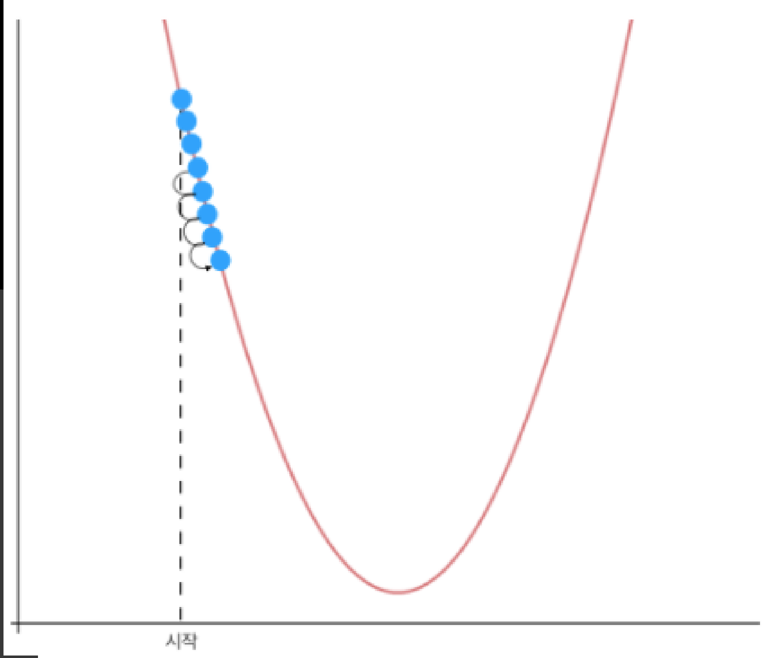


KU-BIG

Gradient Descending & Learning Rate

```
[ ]  eta=0.01
     w=np.zeros((11,1))
     w[0]=5
     print('w(0)=%f' %(w[0]), 'E(w(0))=%f' %(E(w[0])))
     for i in range(0,10):
       w[i+1]=w[i] - eta*dE(w[i])
       print('w(%d)=%f' %(i+1,w[i+1]), 'E(w(%d))=%f' %(i+1,E(w[i+1])))
```

```
w(0)=5.000000 E(w(0))=32.000000
w(1)=4.820000 E(w(1))=28.857200
w(2)=4.650800 E(w(2))=26.080222
w(3)=4.491752 E(w(3))=23.626484
w(4)=4.342247 E(w(4))=21.458361
w(5)=4.201712 E(w(5))=19.542608
w(6)=4.069609 E(w(6))=17.849848
w(7)=3.945433 E(w(7))=16.354126
w(8)=3.828707 E(w(8))=15.032506
w(9)=3.718984 E(w(9))=13.864722
w(10)=3.615845 E(w(10))=12.832869
```



Gradient Descending & Learning Rate

```
[ ]  eta=0.1
     w=np.zeros((11,1))
     w[0]=5
     print('w(0)=%f' %(w[0]), 'E(w(0))=%f' %(E(w[0])))
     for i in range(0,10):
       w[i+1]=w[i] - eta*dE(w[i])
       print('w(%d)=%f' %(i+1,w[i+1]), 'E(w(%d))=%f' %(i+1,E(w[i+1])))
```

```
w(0)=5.000000 E(w(0))=32.000000
w(1)=3.200000 E(w(1))=9.320000
w(2)=2.480000 E(w(2))=5.691200
w(3)=2.192000 E(w(3))=5.110592
w(4)=2.076800 E(w(4))=5.017695
w(5)=2.030720 E(w(5))=5.002831
w(6)=2.012288 E(w(6))=5.000453
w(7)=2.004915 E(w(7))=5.000072
w(8)=2.001966 E(w(8))=5.000012
w(9)=2.000786 E(w(9))=5.000002
w(10)=2.000315 E(w(10))=5.000000
```

Gradient Descending

- Batch Gradient Descent

- Stochastic Gradient Descent

- Mini-Batch Gradient Descent

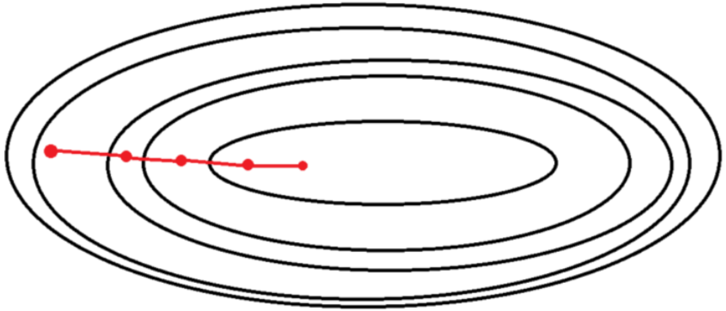
Gradient Descending

- Batch Gradient Descent

- Batch : 한번에 처리하는 데이터의 묶음

- 전체 데이터를 한번에 처리 (1Epoch당 1회 업데이트)

- 항상 같은 데이터를 이용해 학습, 수렴이 안정적



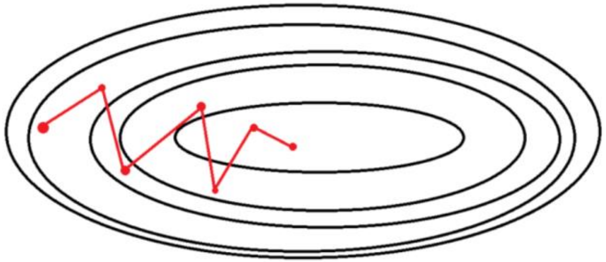
Gradient Descending

- Stochastic Gradient Descent

- 하나의 데이터를 이용하여 경사하강법 진행, 1epoch이 동안 n번 업데이트

- 매번 새로운 데이터를 랜덤으로 고르기 때문에 확률적 경사하강법이라 부름

- 적은 데이터로 학습가능, 속도 빠름, but Shooting 발생(Local minima에 빠질 가능성을 줄여주기도 함)



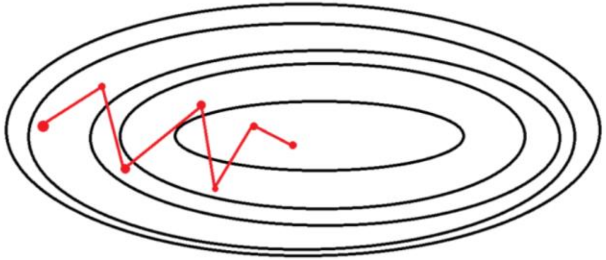
Gradient Descending

- Mini-Batch Gradient Descent

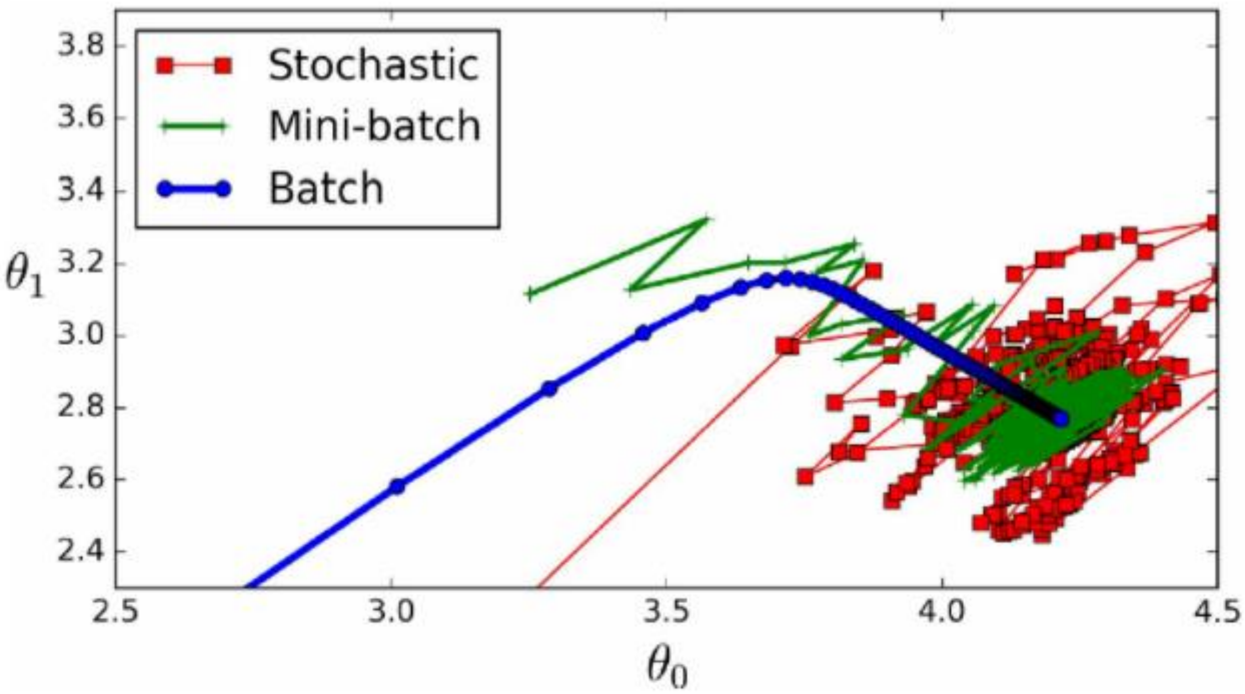
\- 하나의 데이터를 이용하여 경사하강법 진행, 1epoch이 동안 n번 업데이트

\- 매번 새로운 데이터를 랜덤으로 고르기 때문에 확률적 경사하강법이라 부름

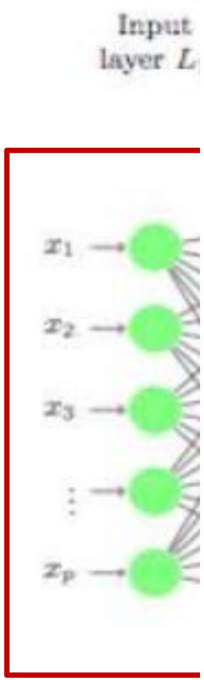
\- 적은 데이터로 학습가능, 속도 빠름, but Shooting 발생(Local minima에 빠질 가능성을 줄여주기도 함)

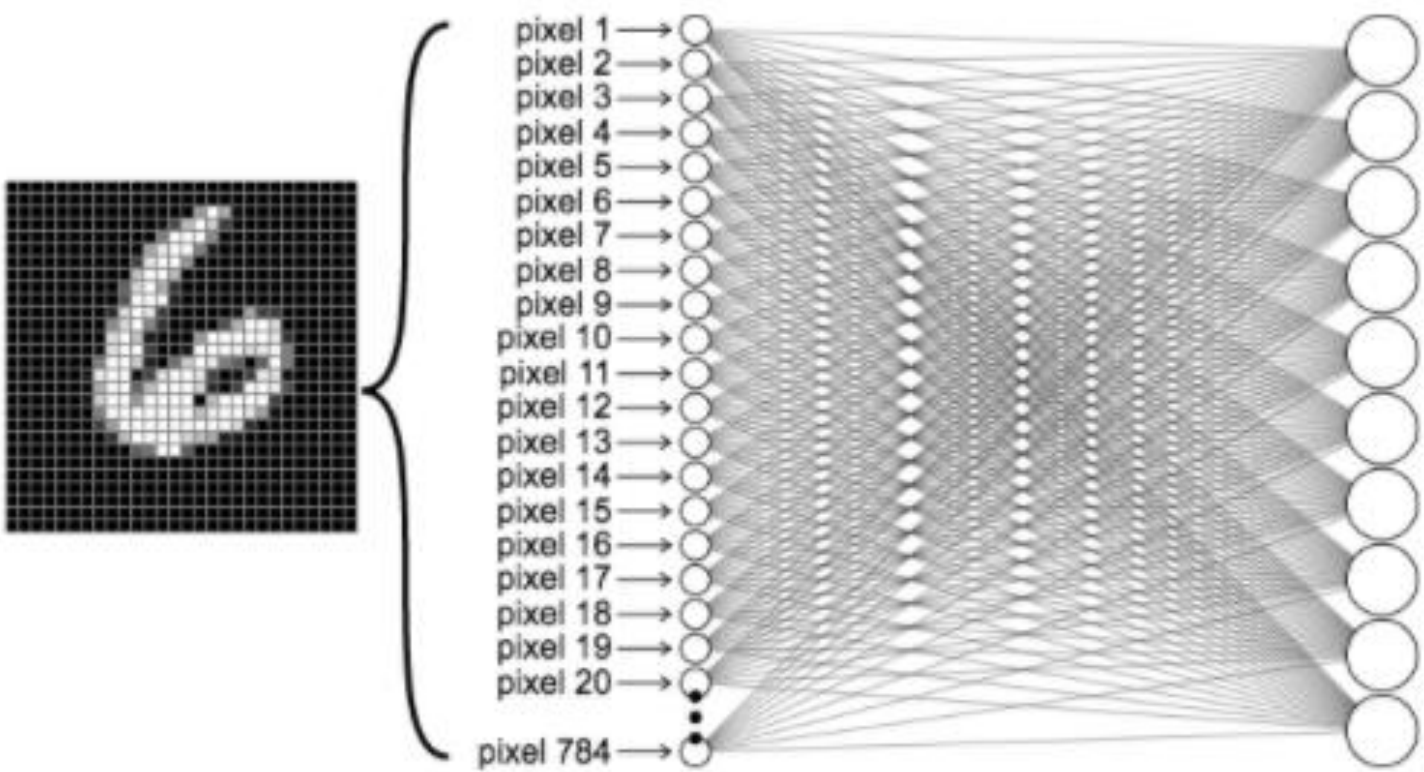


Gradient Descending



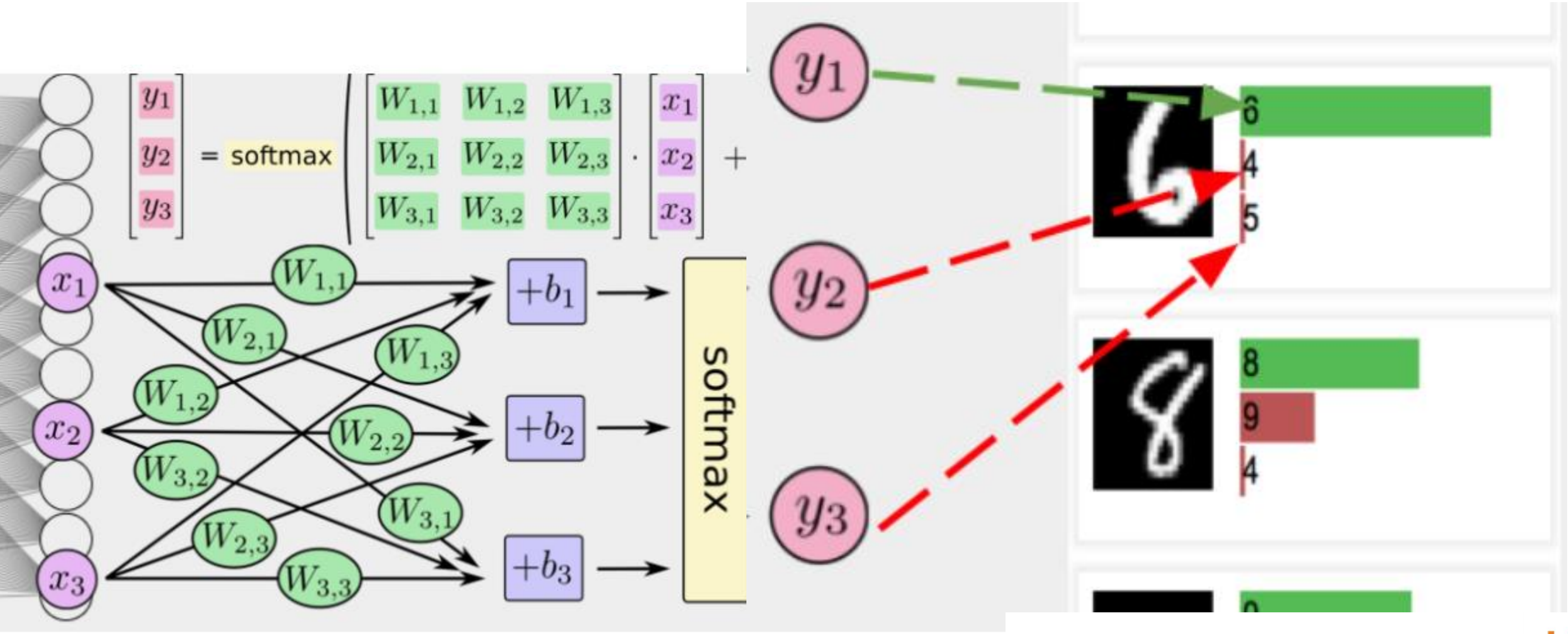
Multi-Layer Perceptron





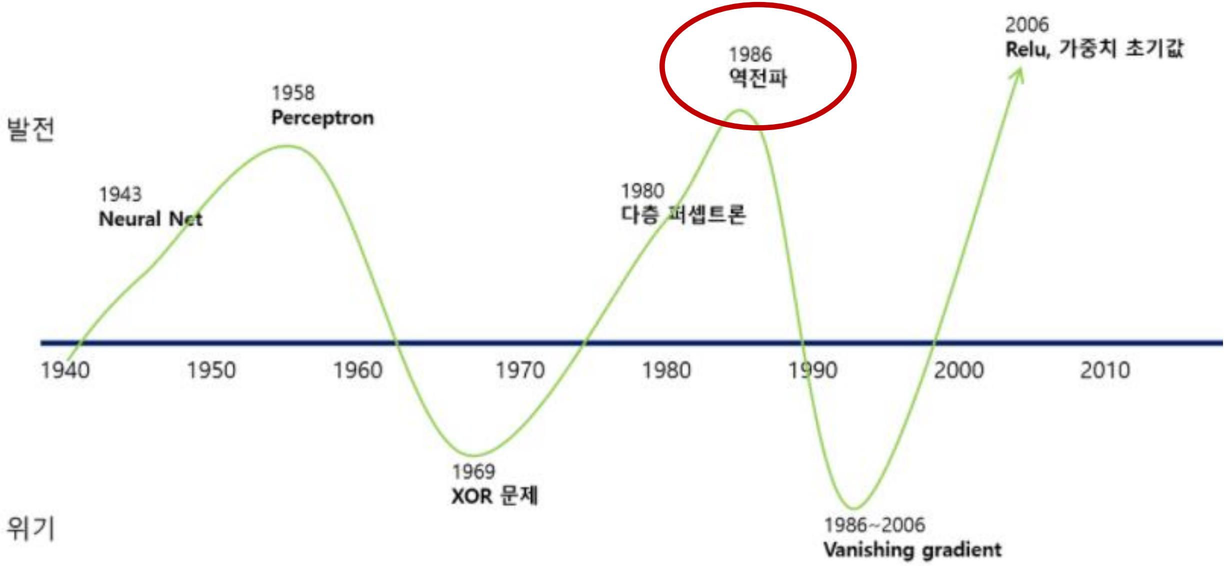
28x28=784

Multi-Layer Perceptron



Backpropagation

- 어떻게 weights를 수정할 것인가?

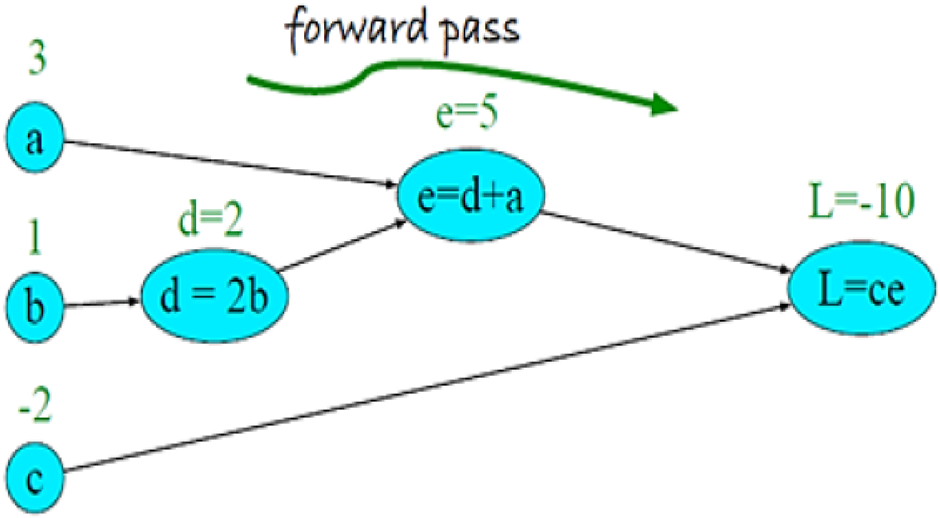


Backpropagation

- Computation Graph

$$L(a,b,c) = c(a+2b).$$

$$d = 2*b$$
$$e = a+d$$
$$L = c*e$$



Backpropagation

- Chain Rule

$$L(a,b,c) = c(a+2b).$$

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$

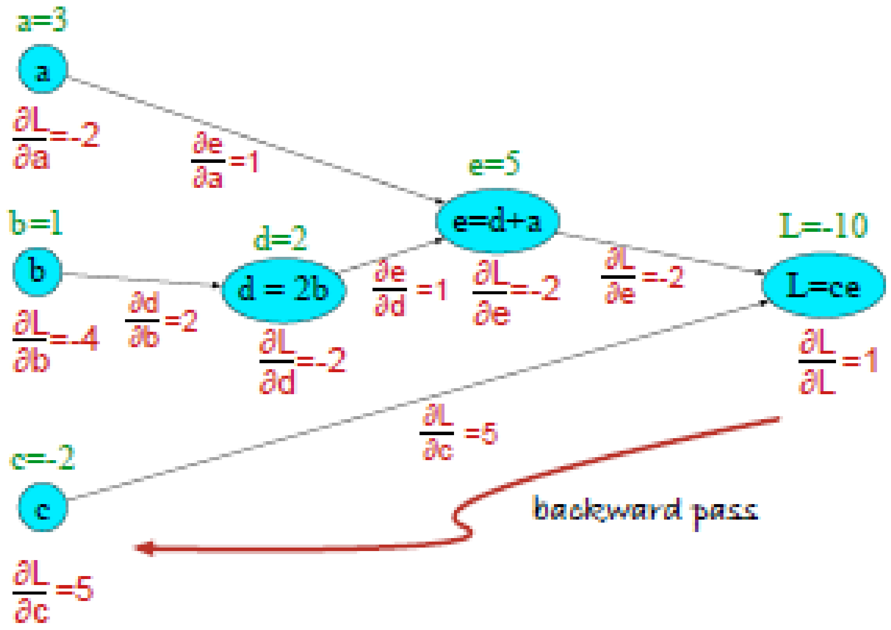
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$\frac{\partial L}{\partial c} = e$$

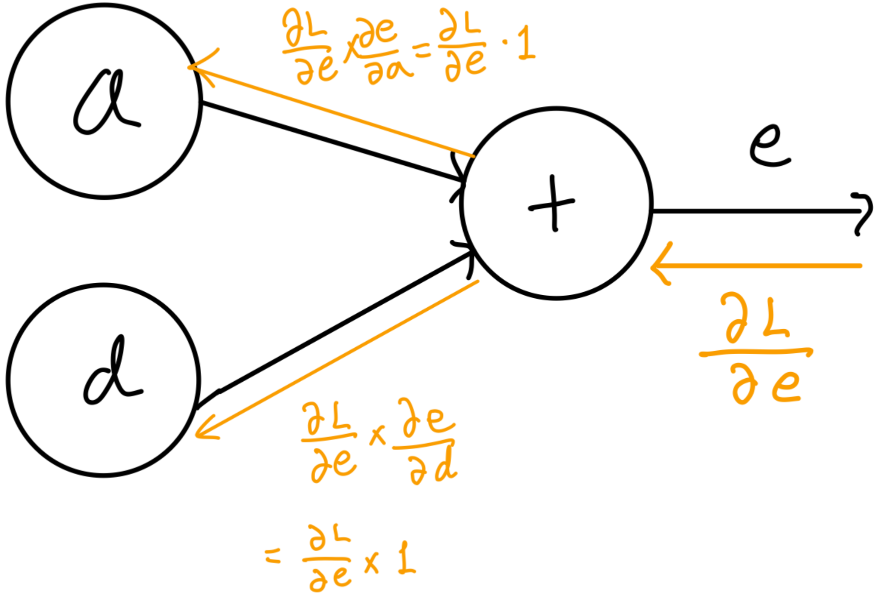
Backpropagation

- Back Propagation



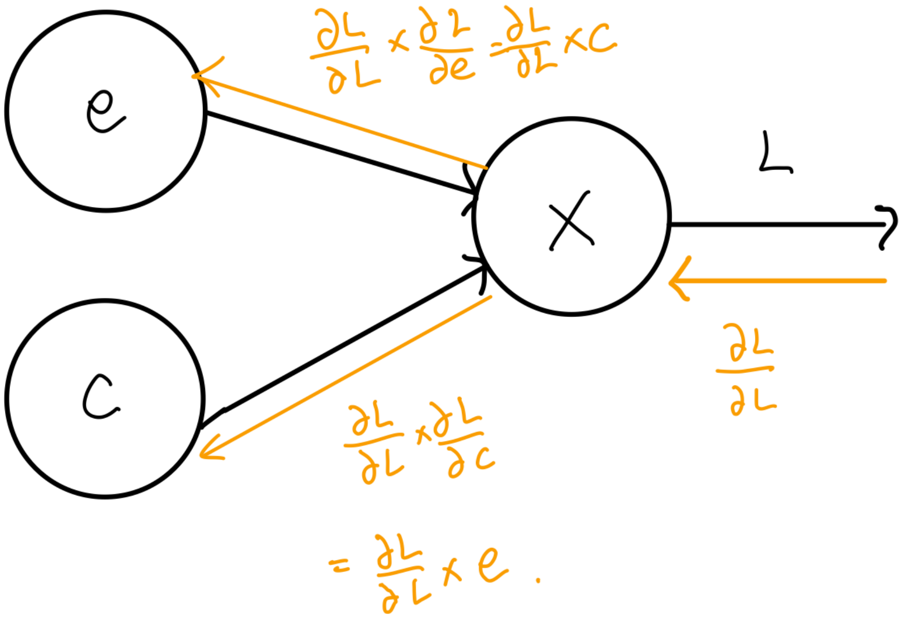
Backpropagation

- 덧셈 노드의 역전파



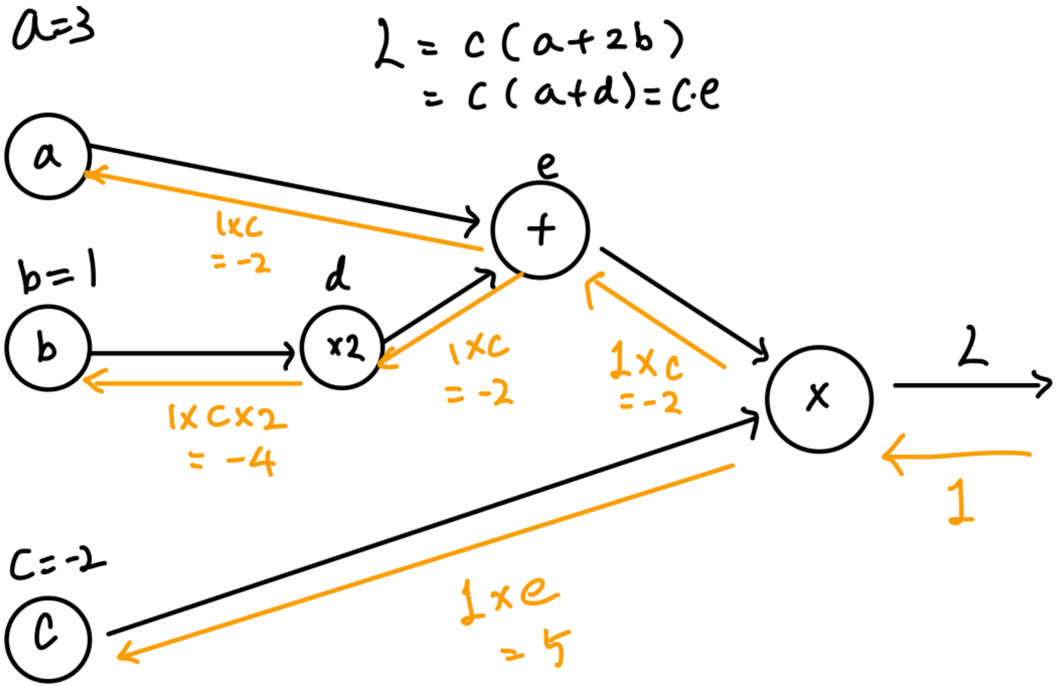
Backpropagation

- 곱셈 노드의 역전파



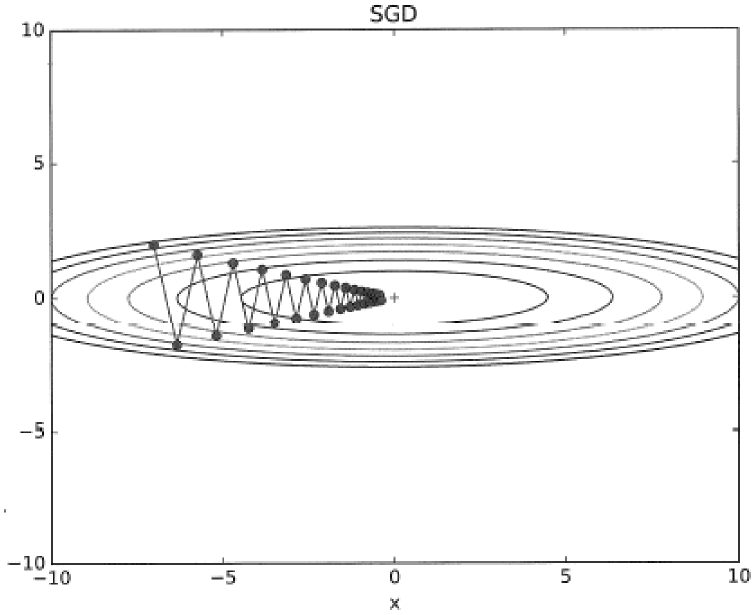
Backpropagation

- 예



Optimizer

- Stochastic Gradient Descent

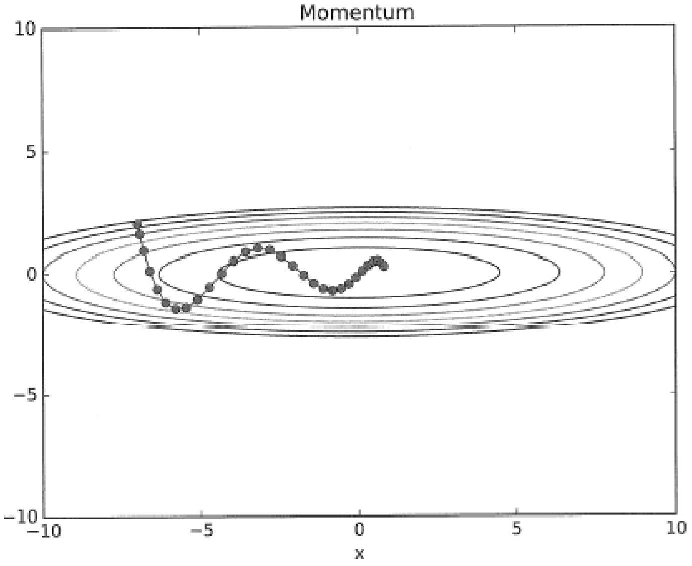


Optimizer

- Momentum

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

$$\mathbf{W} \leftarrow \mathbf{W} + \mathbf{v}$$

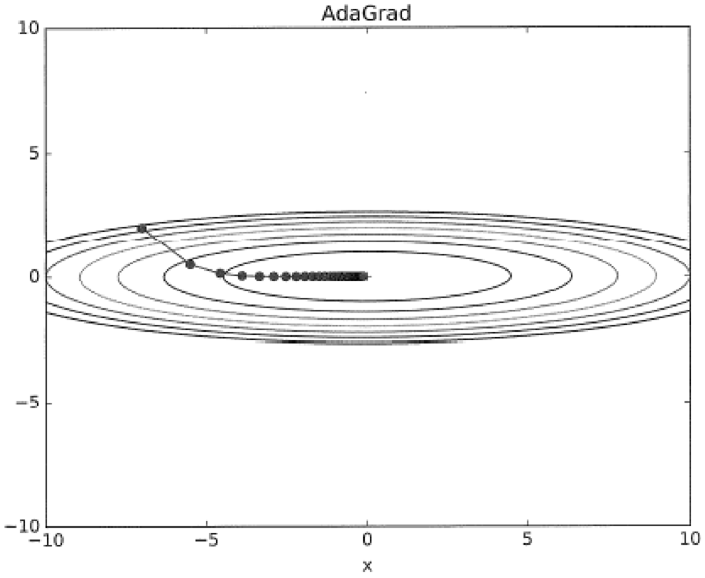


Optimizer

- AdaGrad

$$\mathbf{h} \leftarrow \mathbf{h} + \frac{\partial L}{\partial \mathbf{W}} \odot \frac{\partial L}{\partial \mathbf{W}}$$

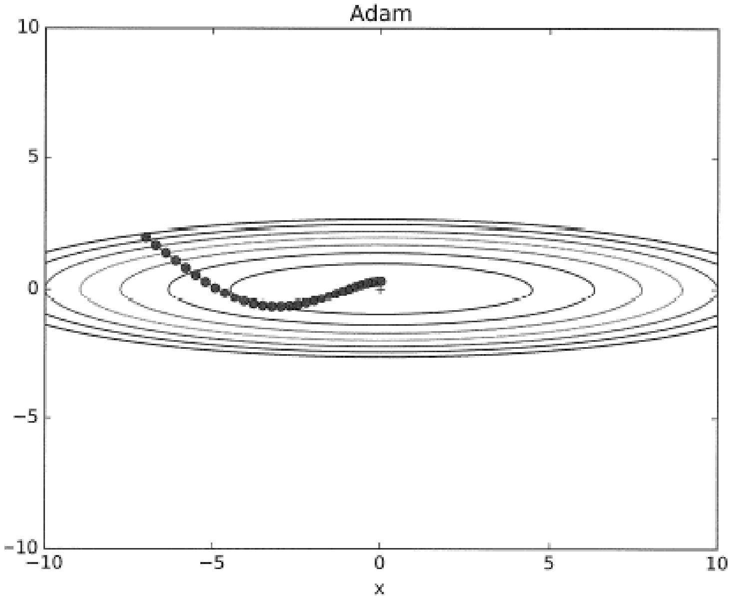
$$\mathbf{W} \leftarrow \mathbf{W} + \eta \frac{1}{\sqrt{\mathbf{h}}} \frac{\partial L}{\partial \mathbf{W}}$$



Optimizer

- Adam

AdaGrad + Momentum

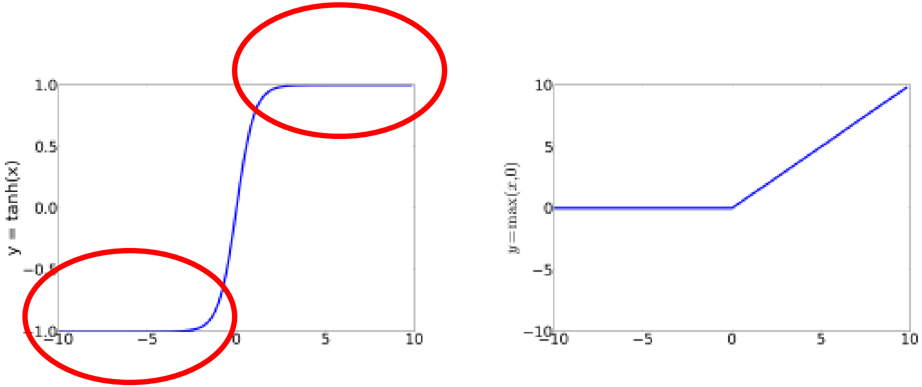


Weight Initialization

- 가중치 최소화 -> 오버피팅 방지법 중 하나
- 최소화 하려면 처음부터 0으로 두면 안될까? -> 역전파 -> 노드를 여러 개 만든 의미가 사라지게 된다.
- 초기 가중치를 랜덤하게 설정. (분포를 이용한 난수추출)

Weight Initialization

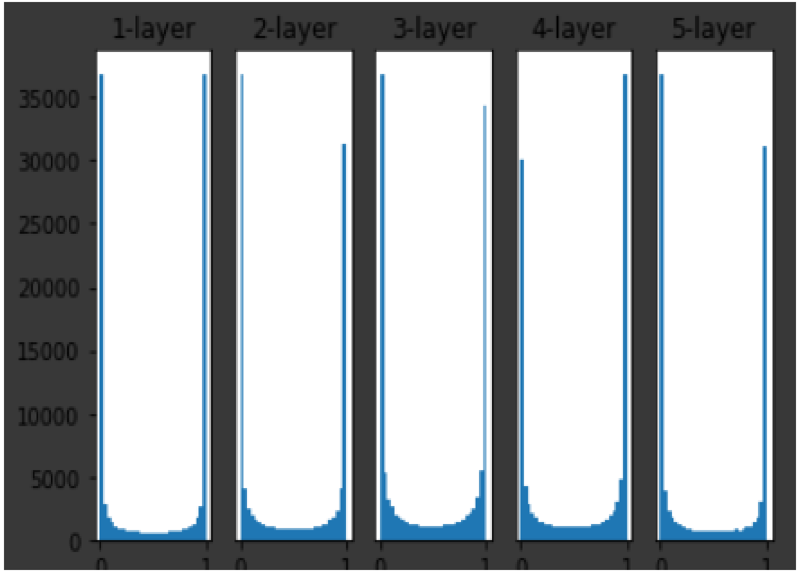
- Vanishing Gradient



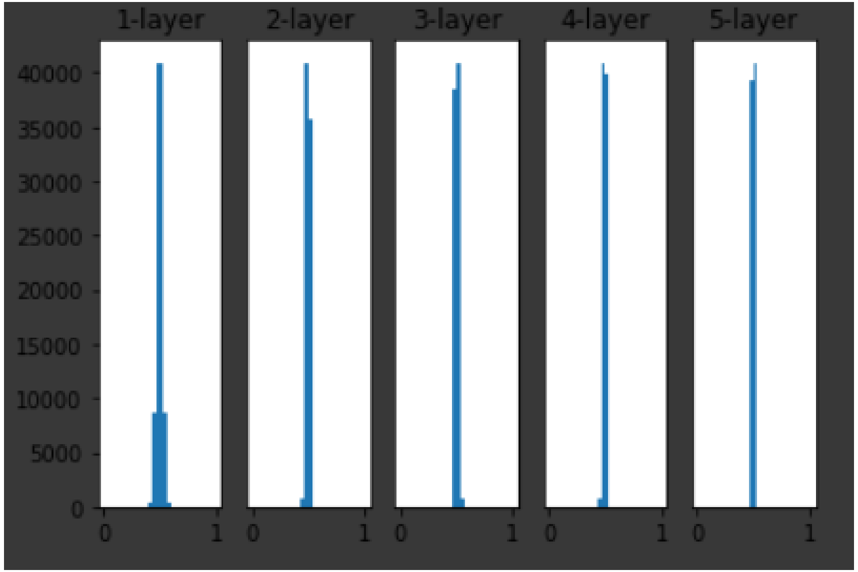
Weight Initialization

- Sigmoid

$N(0, 1^2)$



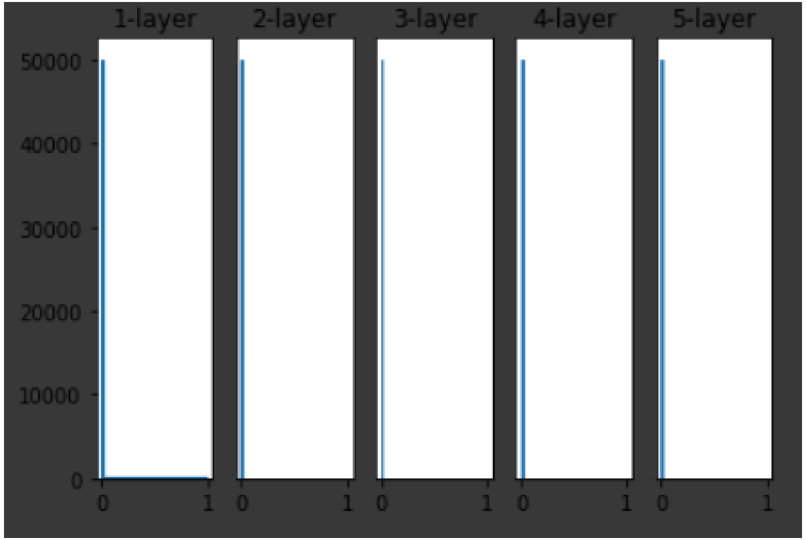
$N(0, (0.01)^2)$



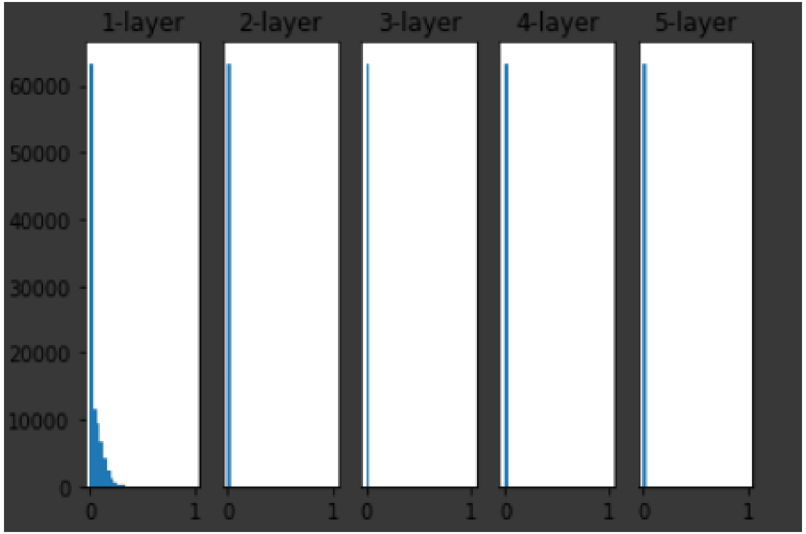
Weight Initialization

- ReLU

$N(0, 1^2)$

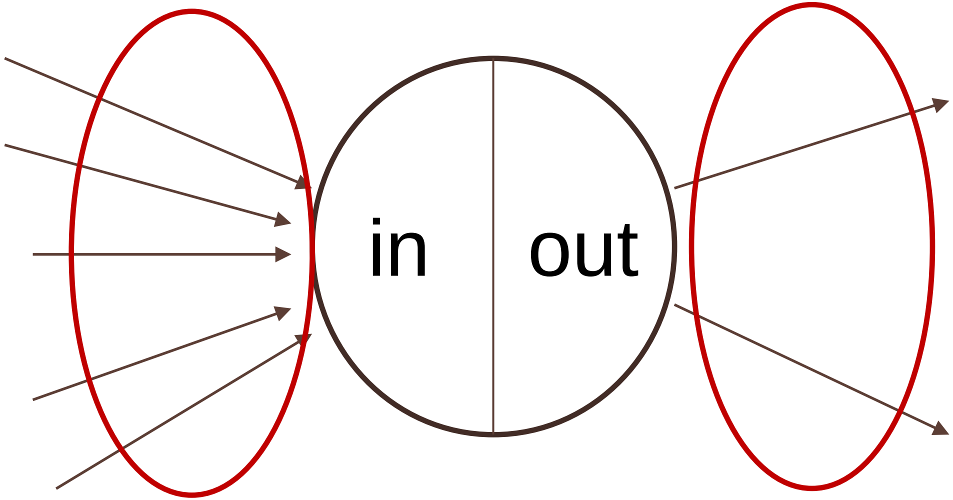


$N(0, (0.01)^2)$



Weight Initialization

- Xavier Initialization



n_{in} n_{out}

Weight Initialization

- Xavier Initialization

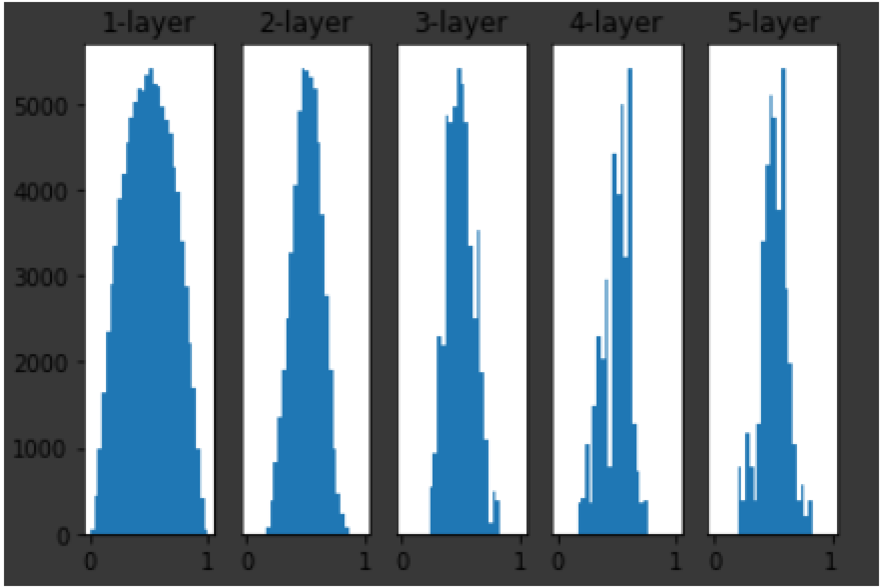
$$W \sim Unif\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}}\right)$$

$$W \sim N(0, \left(\sqrt{\frac{2}{n_{in}+n_{out}}}\right)^2)$$

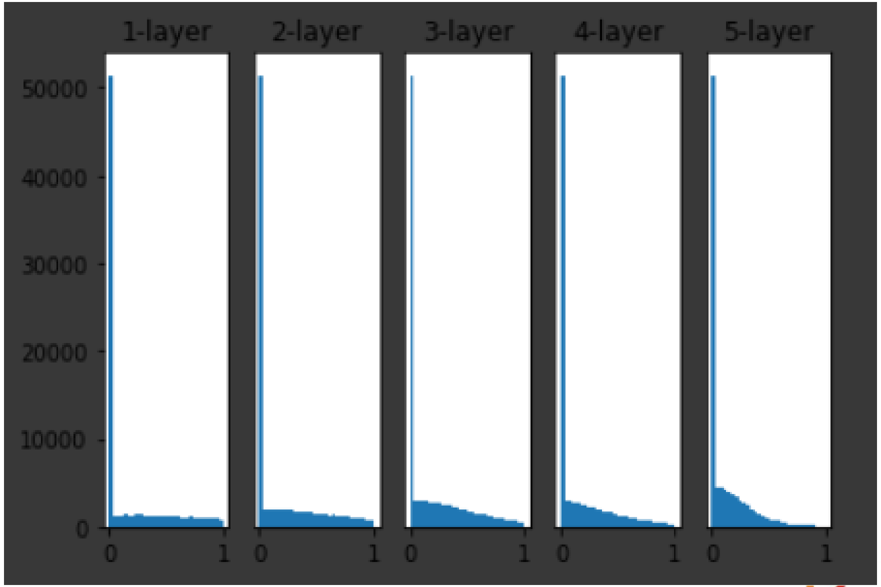
Weight Initialization

- Xavier Initialization

Sigmoid



ReLU



KU-BIG

Weight Initialization

- He Initialization

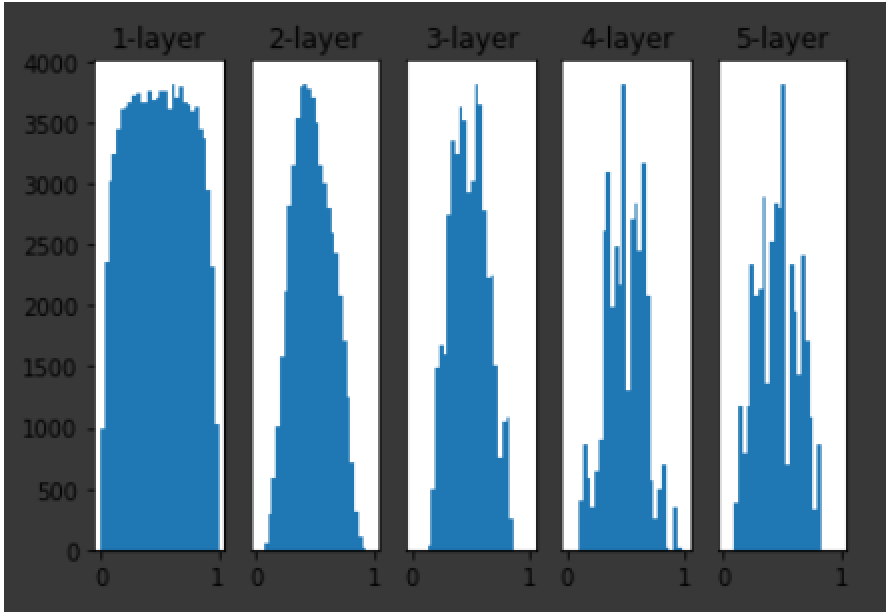
$$W \sim Unif\left(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}}\right)$$

$$W \sim N(0, \left(\sqrt{\frac{2}{n_{in}}}\right)^2)$$

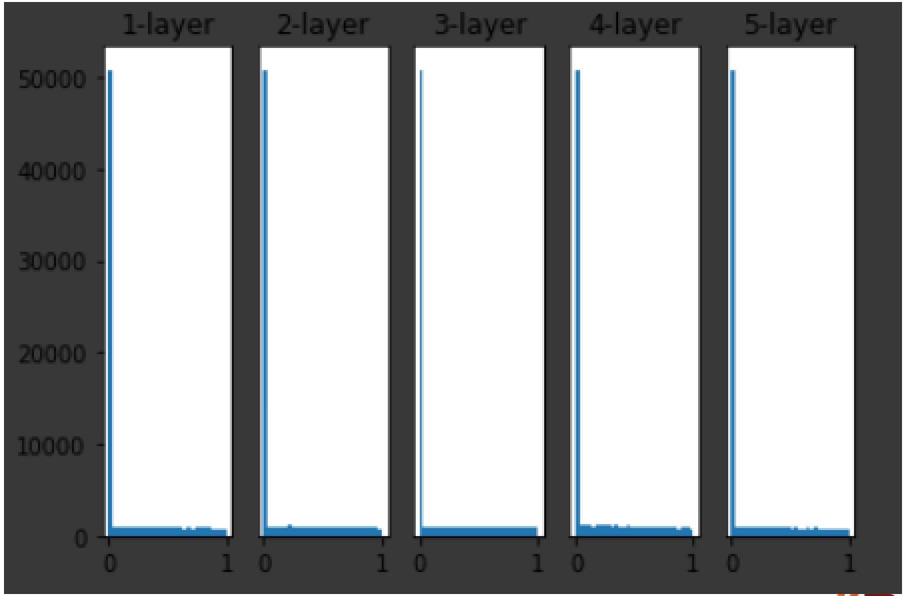
Weight Initialization

- He Initialization

Sigmoid



ReLU



KU-BIG

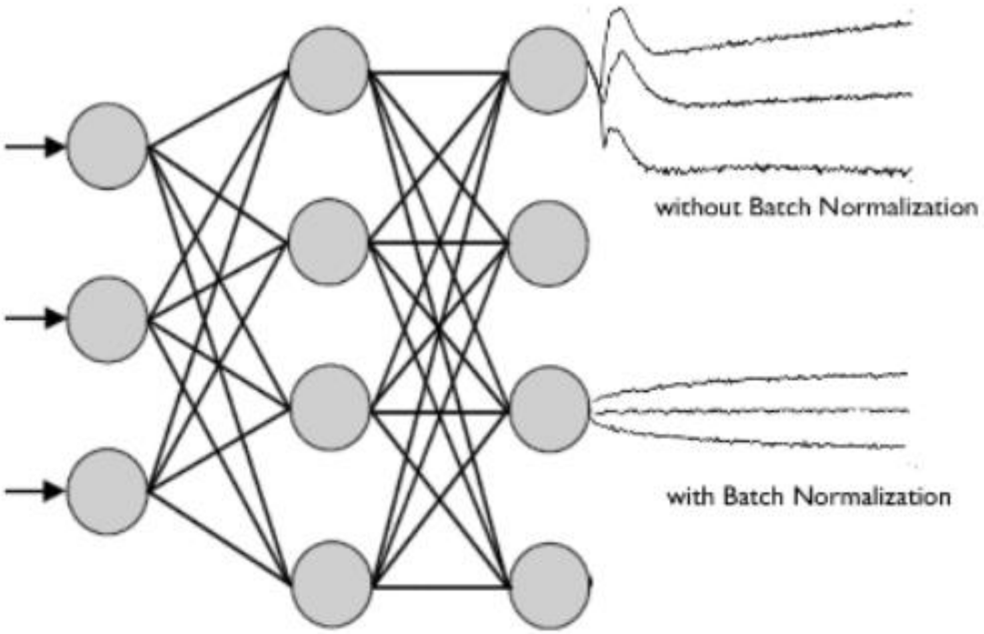
Batch Normalization

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



KU-BIG

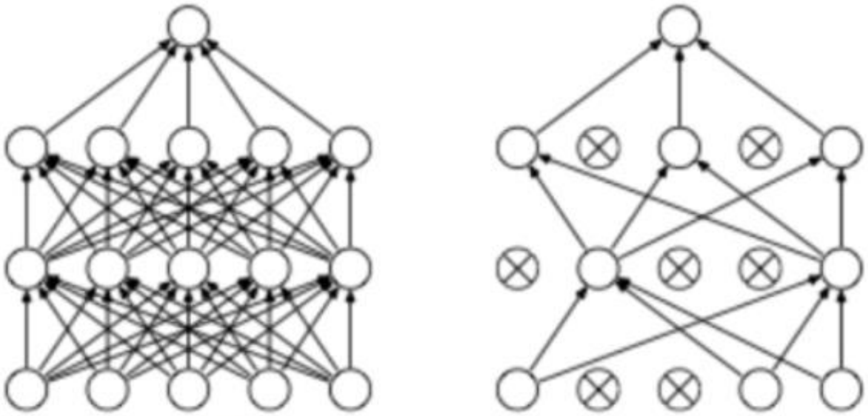
OverFitting

- 너무 많은 매개변수 (복잡한 모델)
- 적은 훈련데이터

Dropout

- Dropout

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]



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

Regularization

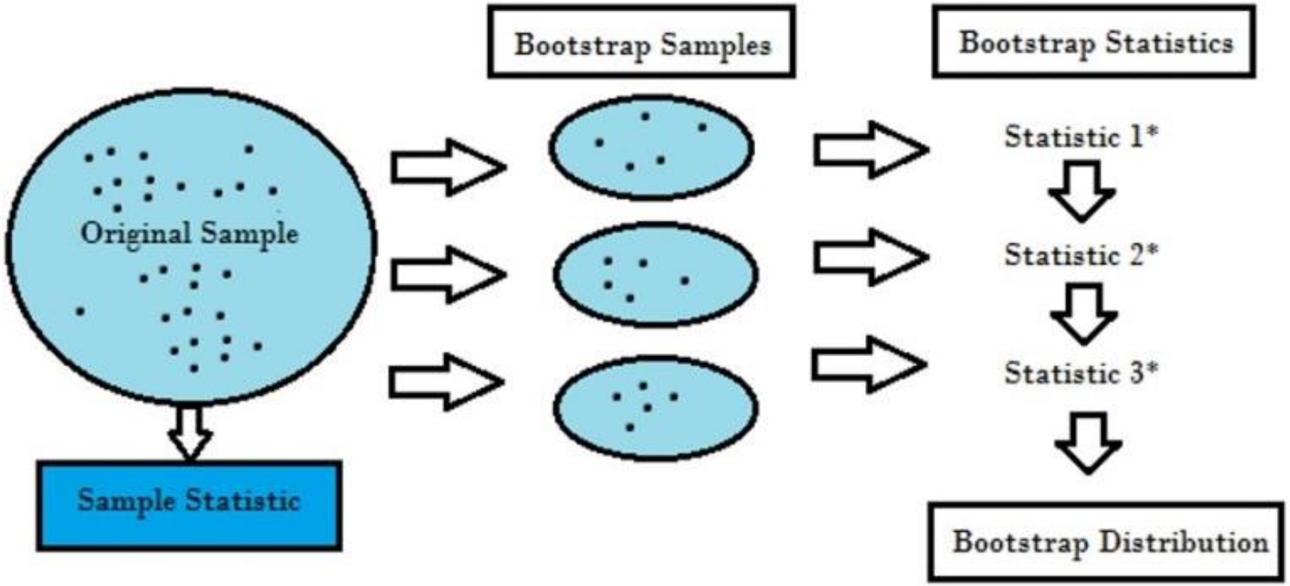
- L1, L2

$$\text{L1}: Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \widehat{y_i}) + \lambda|w|\}, (0 < \lambda < 1)$$

$$\text{L2}: Cost = \frac{1}{n}\sum_{i=1}^{n}\{L(y_i, \widehat{y_i}) + \lambda(w)^2\}, (0 < \lambda < 1)$$

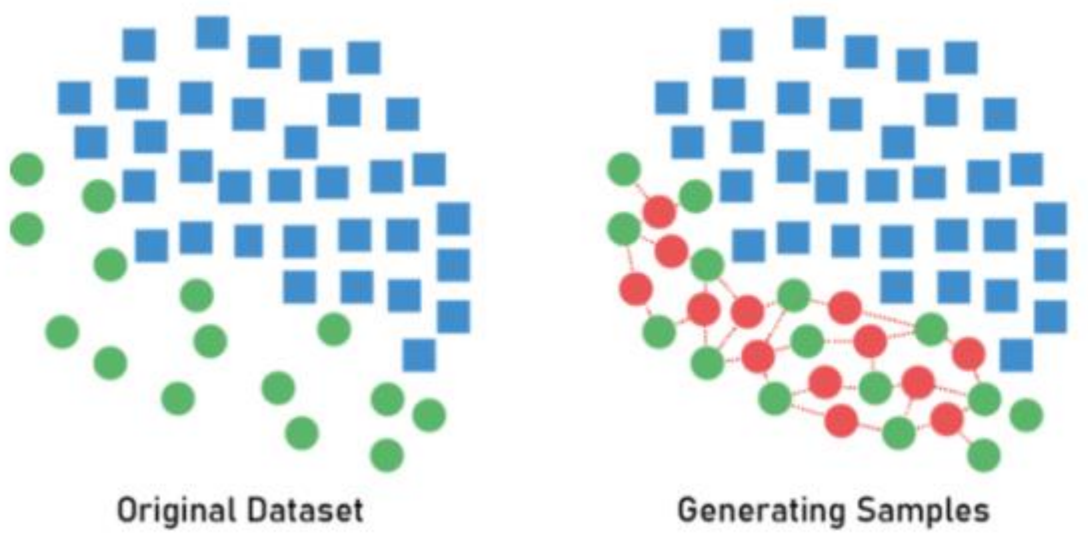
Bootstrap

- Bootstrap



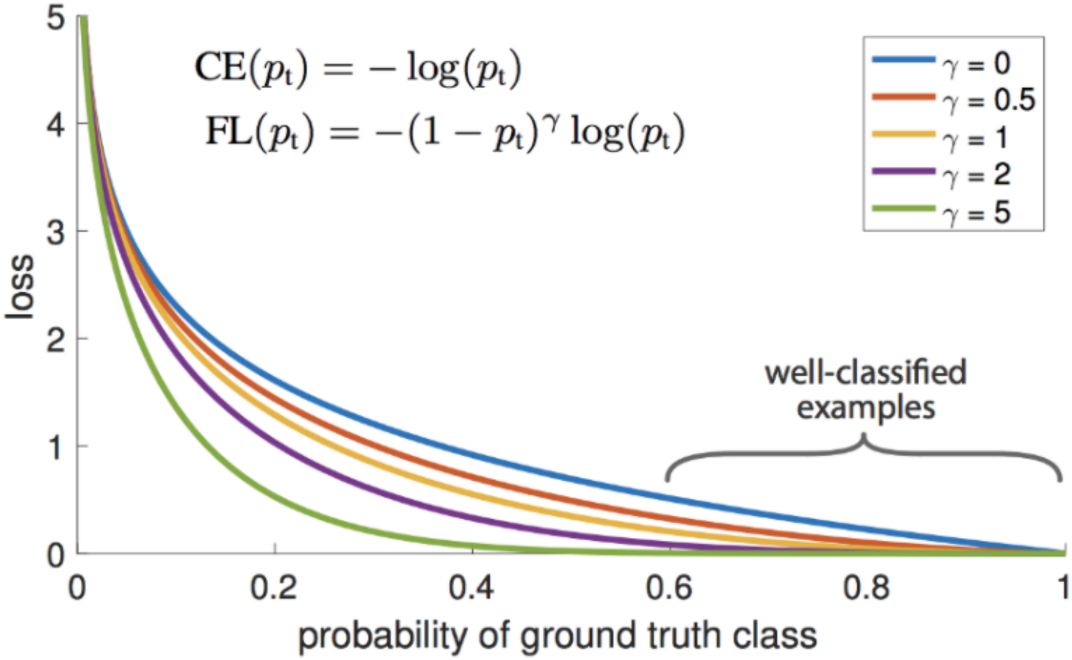
SMOTE

- SMOTE



Focal Loss

- Focal Loss



Hyper-parameter Optimization

- **0단계**
  하이퍼파라미터 값의 범위를 설정한다.
- **1단계**
  설정된 범위에서 하이퍼파라미터의 값을 무작위로 추출한다.
- **2단계**
  1단계에서 샘플링한 하이퍼파라미터 값을 사용하여 학습하고, 검증 데이터로 정확도를 평가한다(단, 에폭은 작게 설정한다).
- **3단계**
  1단계와 2단계를 특정 횟수(100회 등) 반복하며, 그 정확도의 결과를 보고 하이퍼파라미터의 범위를 좁힌다.

KU-BIG