AI 세미나



Presented By
윤지원

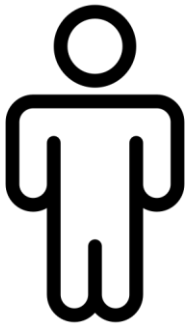
Outline

- AI?
- Task
  - Linear regression
  - Classification
- Model
  - MLP
- Principle of Model
- Optimizer
  - Gradient descent
  - Another optimizer
- Problems & Technic
  - L2 normalization
  - Batch normalization
- CNN
- My experiments

AI

- AI란?
  - 인간의 학습능력, 추론능력, 지각능력을 인공적으로 구현





인간

개



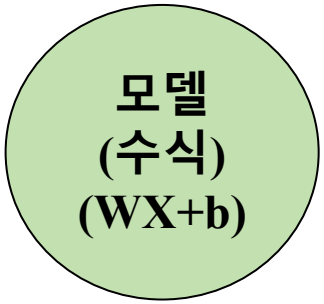
모델
(수식)
(WX+b)

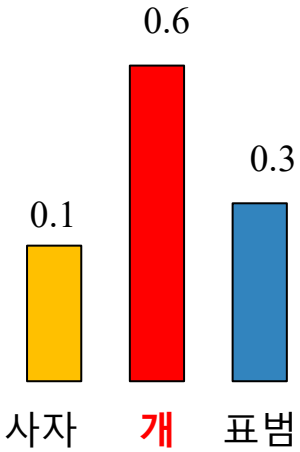
개(0.63)

- AI란?
  - AI 모델은 수식을 통해 정보를 처리하는 구조
    - 모든 입력과 출력을 수치로 다뤄야 함



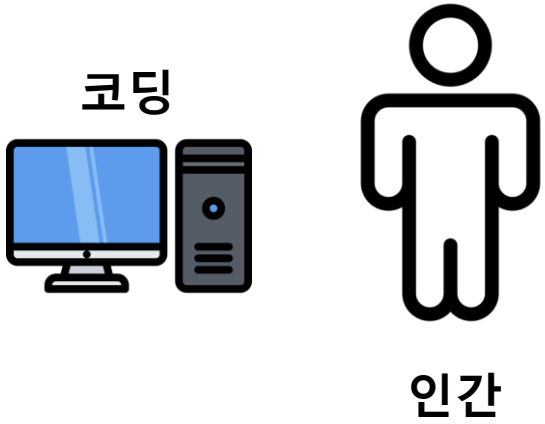
0	**0**	**0**	**1**	**1**	**0**	**0**
1	1	1	1	1	1	0
0	0	0	1	1	1	0
0	0	1	1	1	1	1
0	0	0	1	0	1	0
0	0	0	0	0	0	0

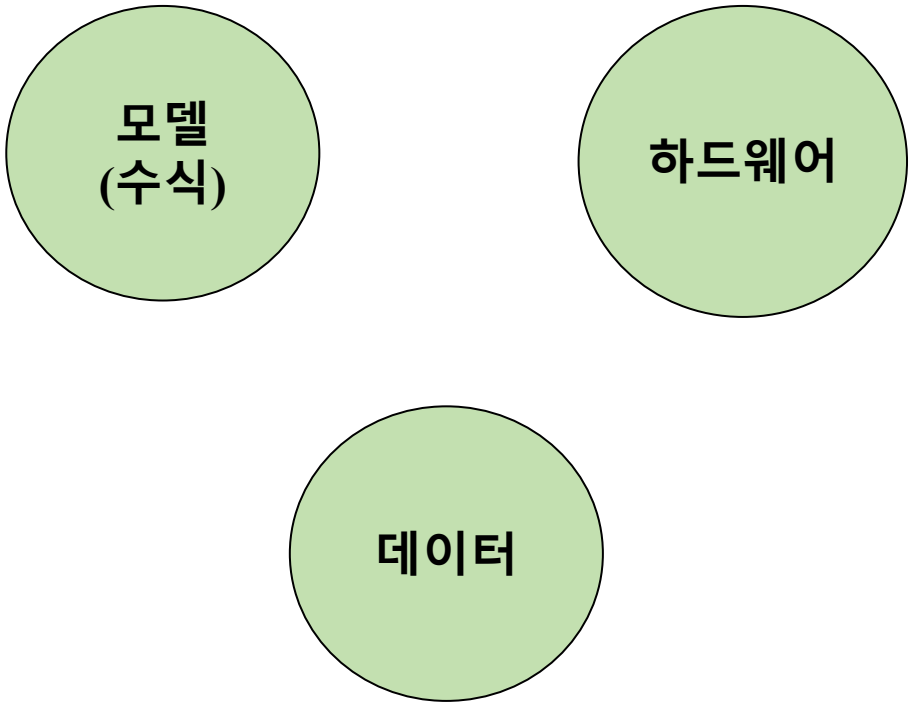




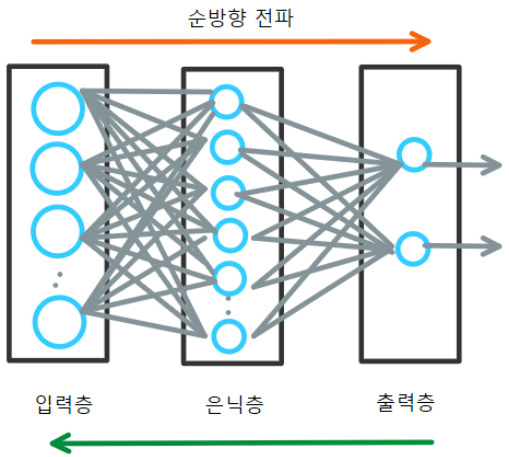
AI

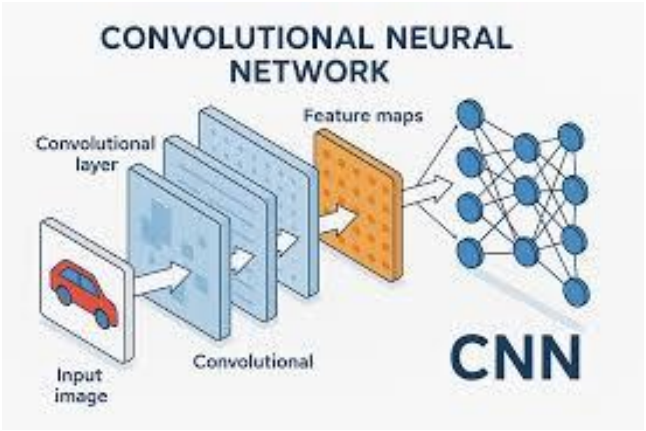
- AI의 3요소

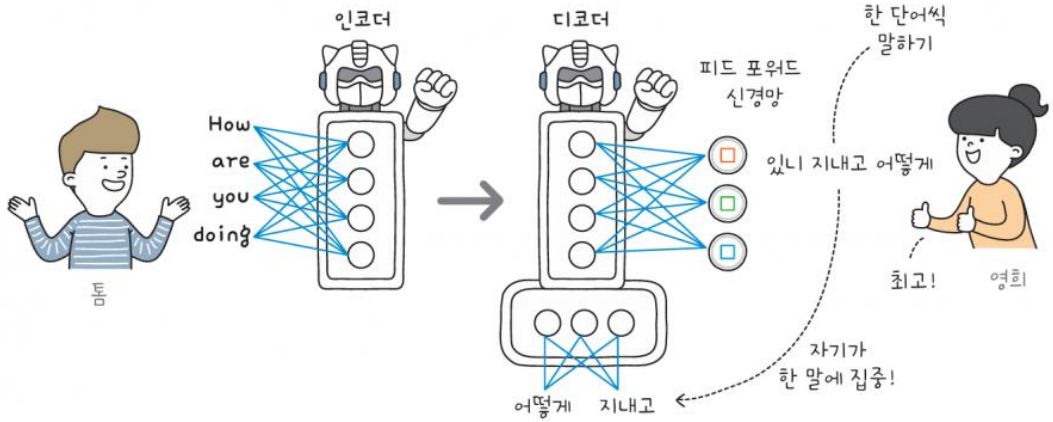




- AI Model
  - 수식의 집합







- Data
  - 모델은 수를 다루므로 수로 표현 해야함



0	0	0	1	1	0	0
1	1	1	1	1	1	0
0	0	0	1	1	1	0
0	0	1	1	1	1	1
0	0	0	1	0	1	0
0	0	0	0	0	0	0

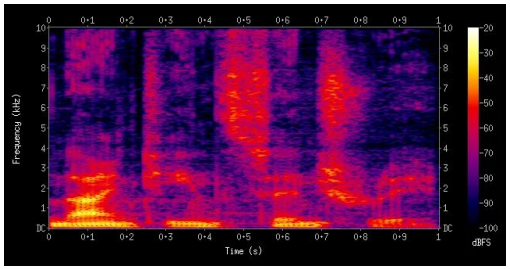
Image

개, 밥, 잠,해 ,달

개 → [0.21, -1.03, 0.77, ...]
밥 → [1.12, 0.54, -0.31, ...]
잠 → [0.98, 0.61, -0.22, ...]
해 → [-0.87, 1.45, 0.12, ...]
달 → [-0.79, 1.38, 0.09, ...]
(예: 768차원 벡터)

Text

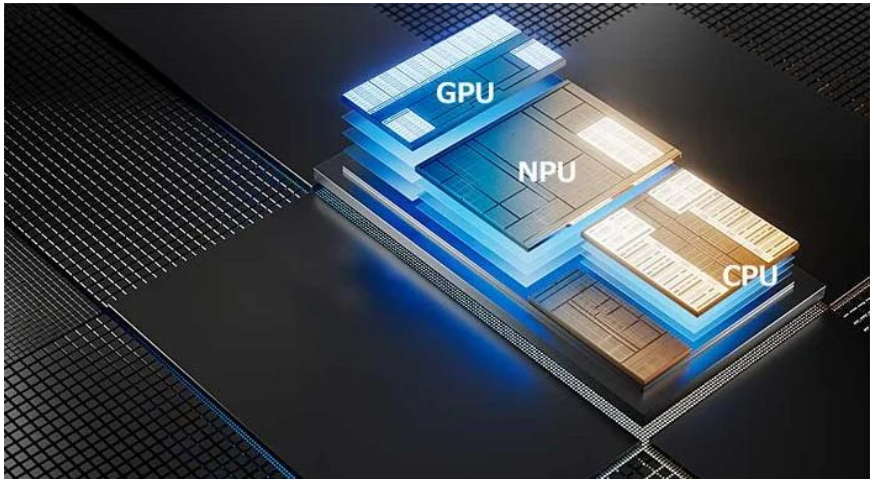




Audio

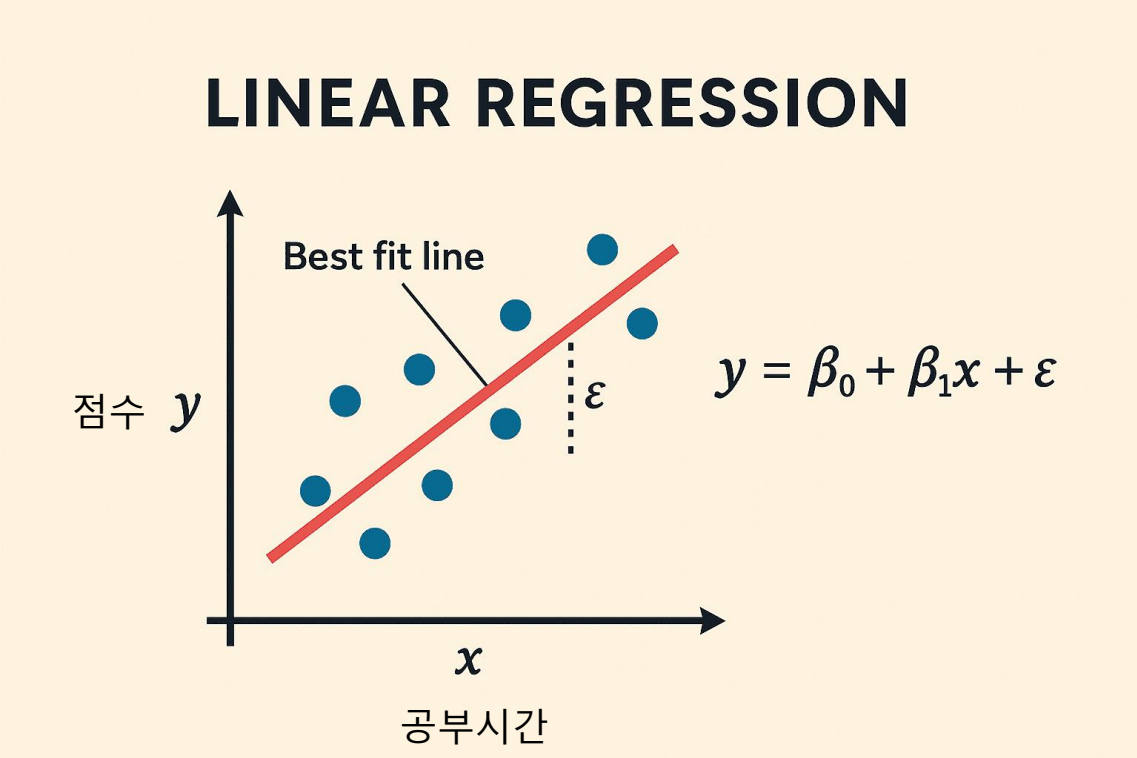
- 하드웨어





AI Task

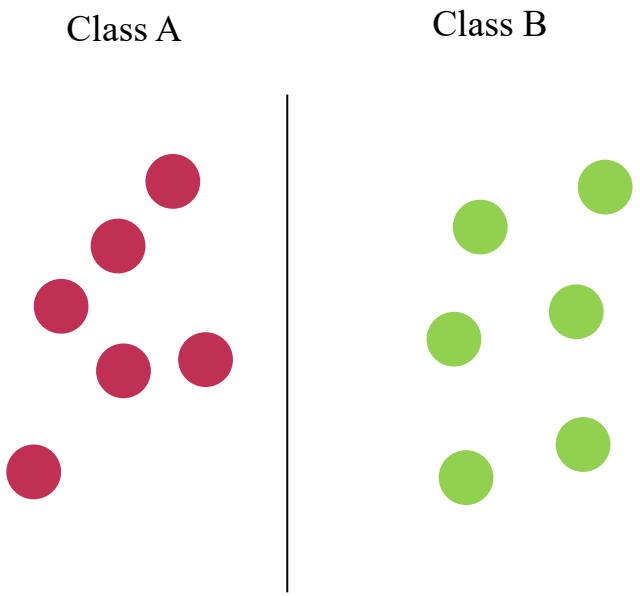
- Linear regression
  - 여러 입력에 대한 출력을 continuous한 값들로 생성
    - 예시: 공부시간, 성적



AI Task

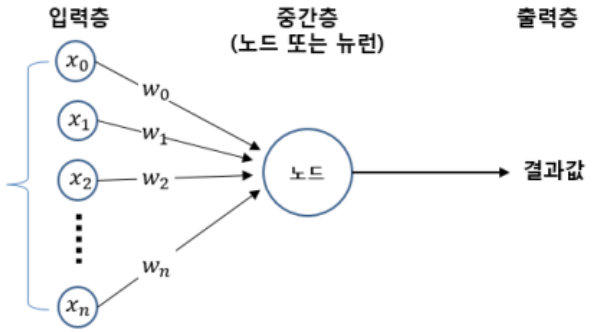
- Classification
  - 여러 입력에 대한 출력을 discrete한 값들로 생성

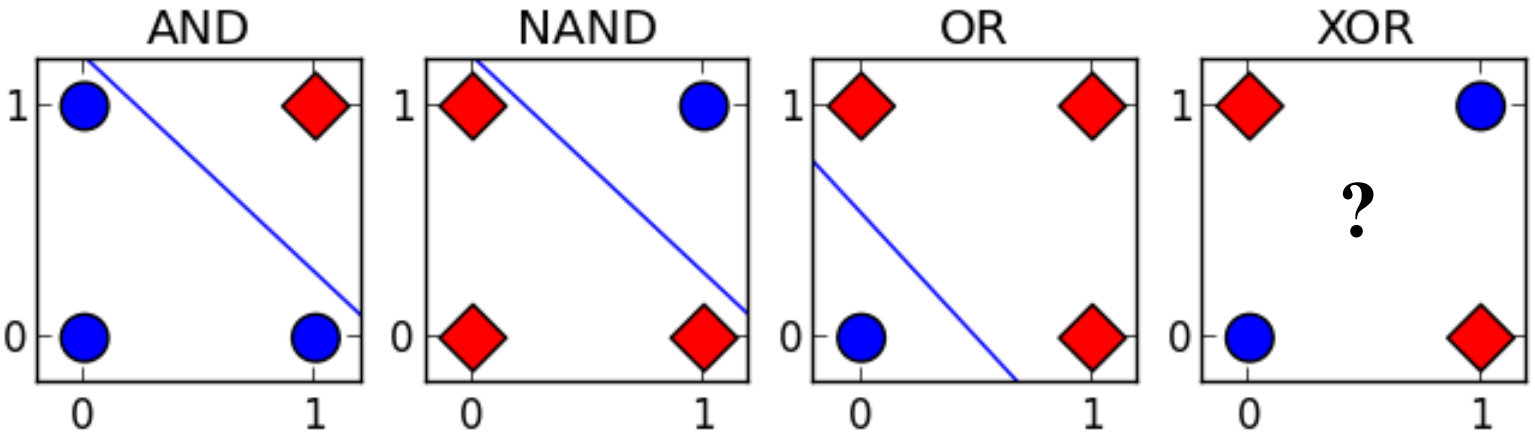




AI Model

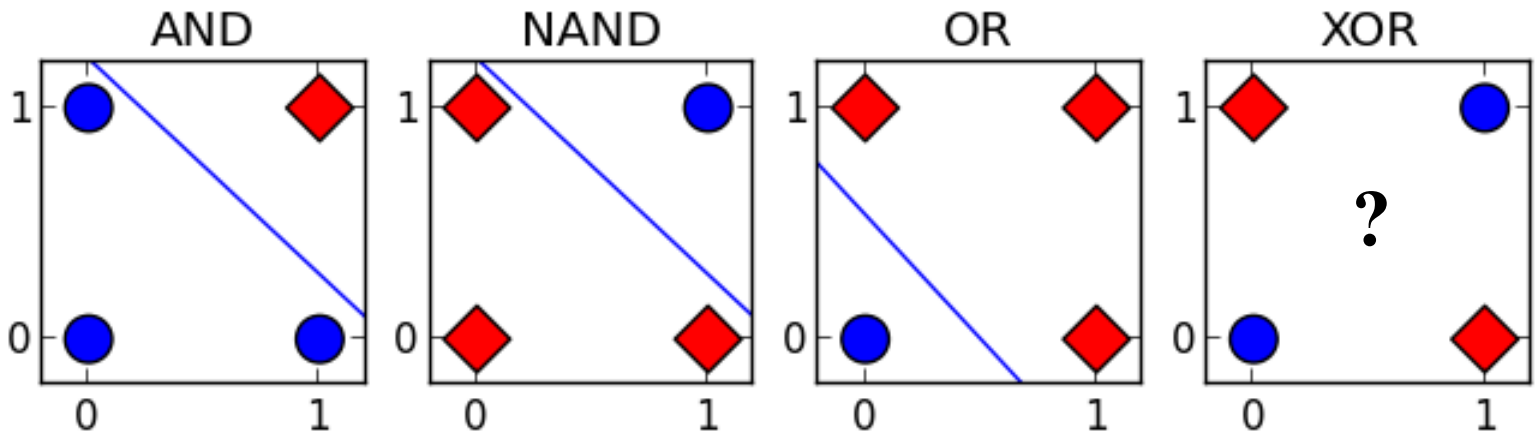
- Background of MLP
  - How to solve **XOR** problem?
    - Use single-layer perceptron ⟶ can't solve





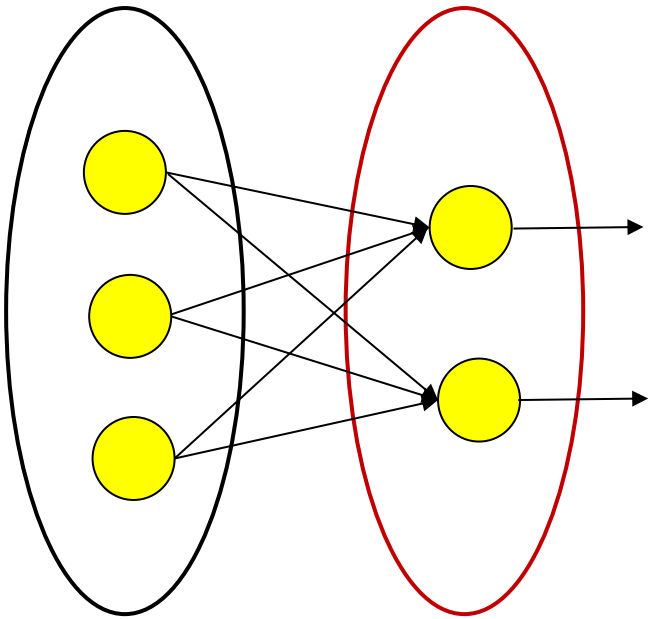
AI Model

- Background of MLP
  - How to solve **XOR** problem?
    - **Sol. 1.** 두개 이상의 선으로 분류



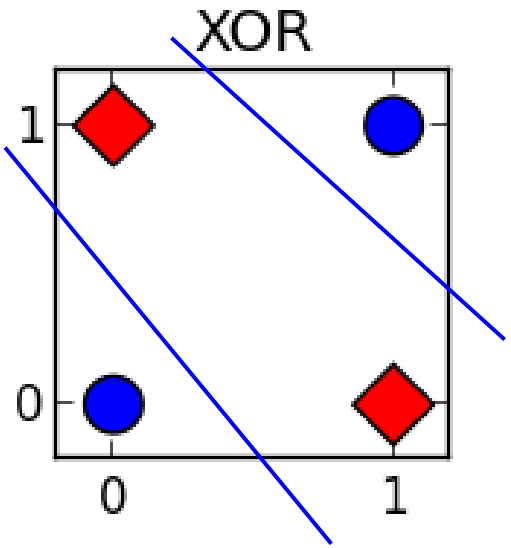
AI Model

- Background of MLP
  - How to solve **XOR** problem?
    - **Sol. 1.** 두개 이상의 선으로 분류



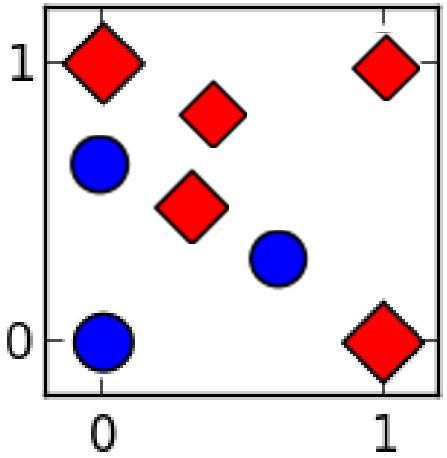
Unit 증가, 위로 길어짐

2개의 선



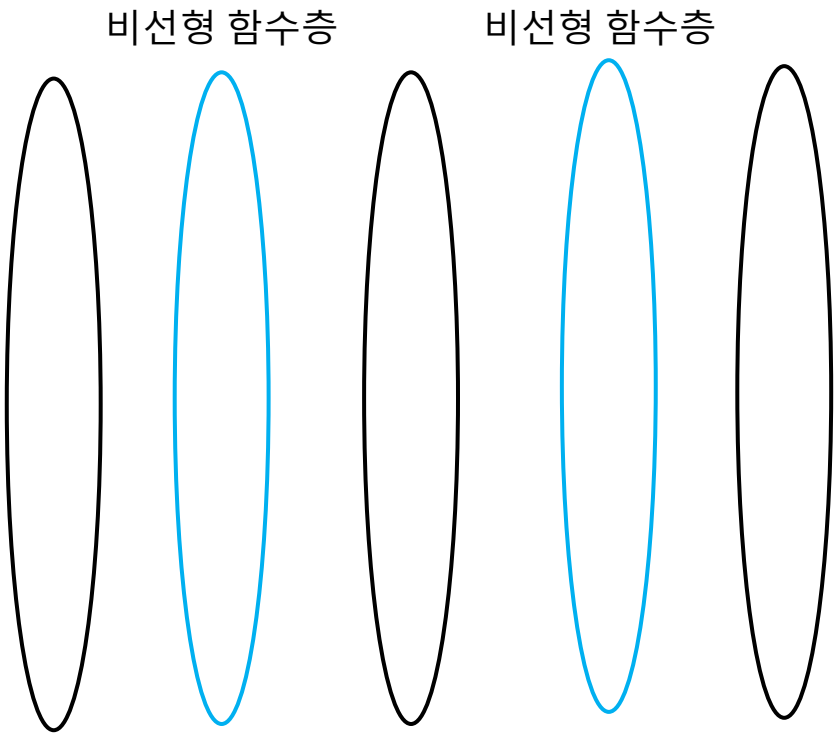
AI Model

- Background of MLP
  - How to solve **This** problem?
    - **Sol. 2.** 곡선으로 분류

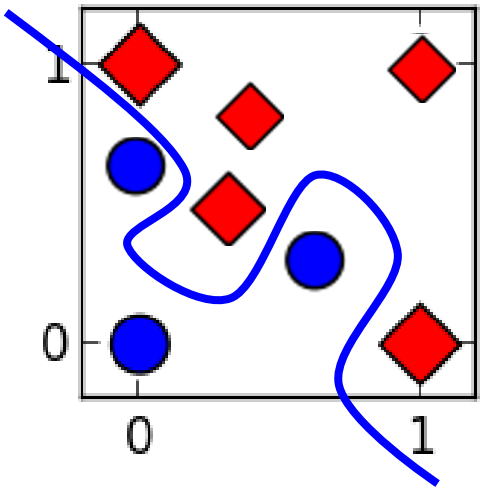


AI Model

- Background of MLP
  - How to solve **This** problem?
    - **Sol. 2.** 곡선으로 분류



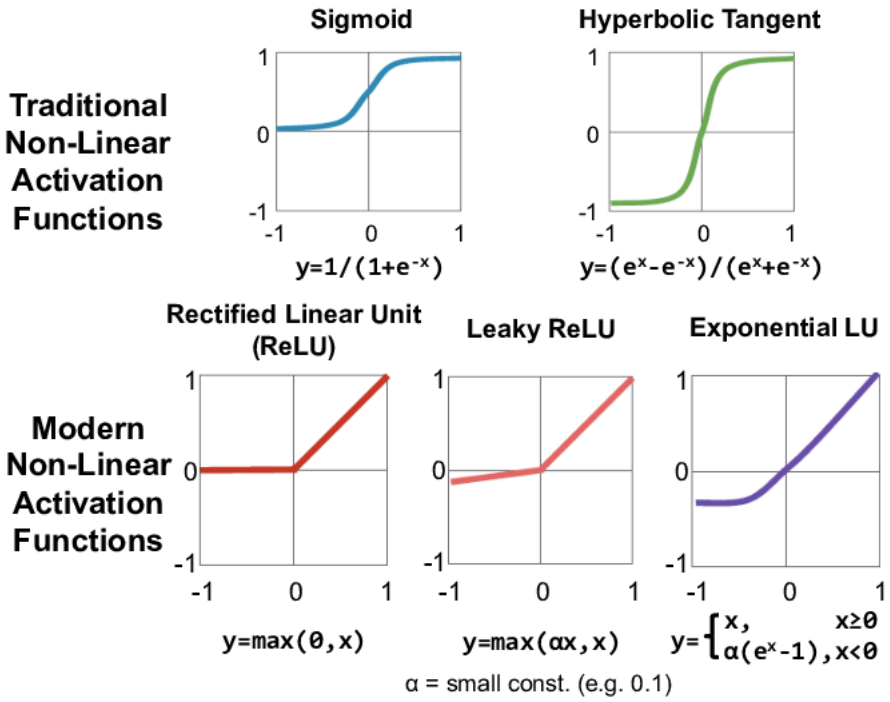
곡선



비선형성 함수층 적용, 옆으로 넓어진다.

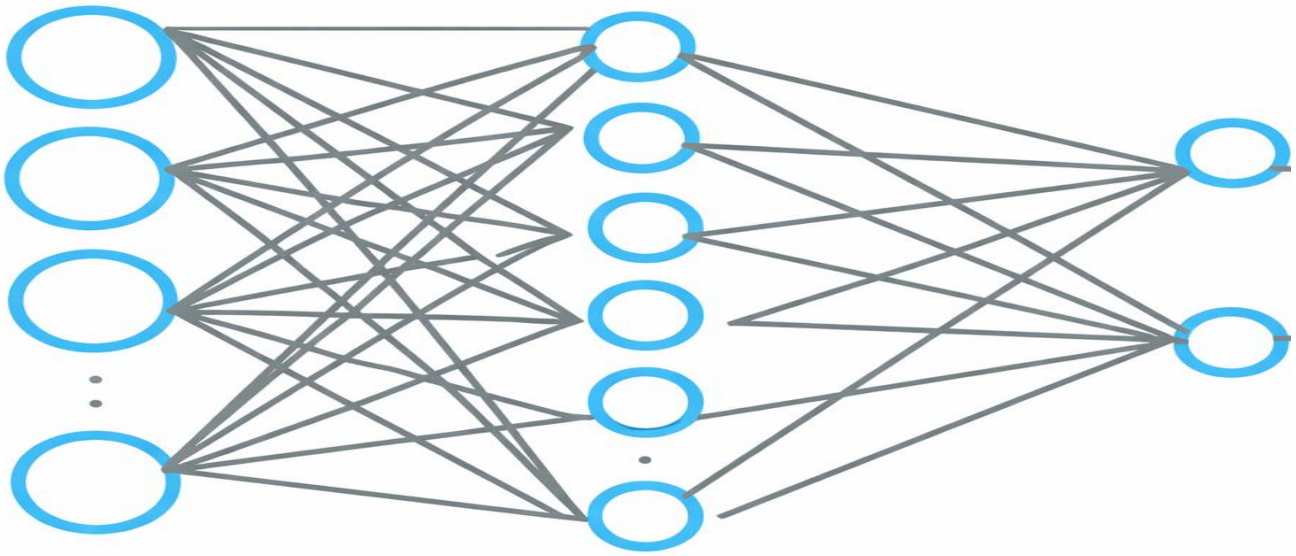
AI Model

- Activation function
  - sigmoid, relu, leaky relu 등



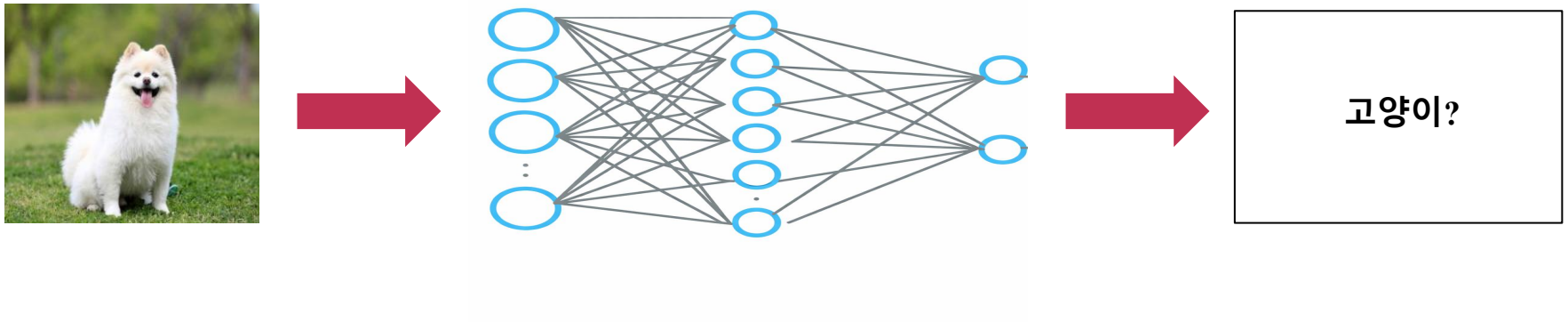
AI Model

- MLP
  - For more complicated problem..
    - 더 많은 선과 곡선이 필요
    - 이를 표현가능한 복잡한 모델이 요구됨
    - **MLP** model 등장



Principle of Model

- How model do task
  - Model의 학습 과정
    - 데이터를 입력 받음
    - 입력으로부터의 출력과 정답사이의 차이를 줄여야 함.
      - ∴ 모델 (weight)를 차이를 줄이는 방향으로 조정

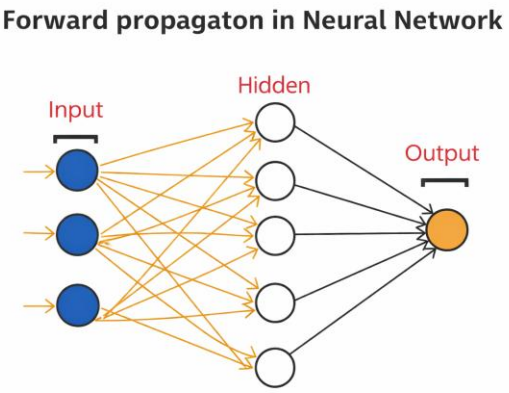


Principle of Model

- Make Output
  - Forward propagation
    - 이미지를 숫자로 변환하여 입력값으로 넣음
    - 입력은 MLP의 input, hiddien, output layer를 거쳐 출력값으로 나옴
    - 출력은 y=wx+b (x가 input, w가 hidden layer)



0	**0**	**0**	**1**	**1**	**0**	**0**
1	1	1	1	1	1	0
0	0	0	1	1	1	0
0	0	1	1	1	1	1
0	0	0	1	0	1	0
0	0	0	0	0	0	0



Principle of Model

- Calculate loss
  - Cross entrophy, softmax in classification
    - 각각의 class에 해당하는 출력의 점수를 확률로 변환
    - 그중 정답 class에 해당하는 확률을 cross entrophy에 적용하면 loss값 출력됨
    - 확률이 낮을 수록 loss가 증가

$$\sigma(z_i) = softmax(z_i) = \frac{e^{z_i}}{e^{z_1}+e^{z_2}\cdots e^{z_n}} = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

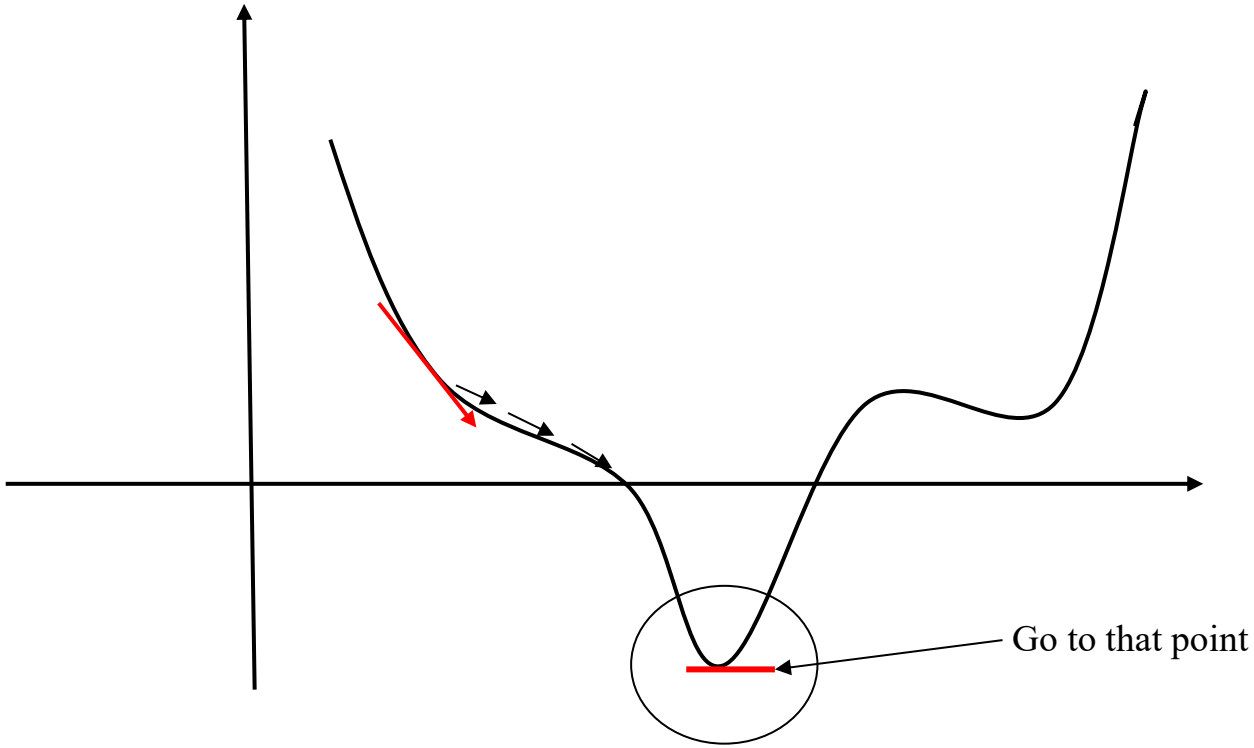
$$H_p(q) = -\sum_{i=1}^{n} q(x_i)\log p(x_i)$$

  - MSE in regression
    - 출력값과 정답값의 차이를 제곱해서 모두 더하고 그것의 평균을 loss로 취함

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

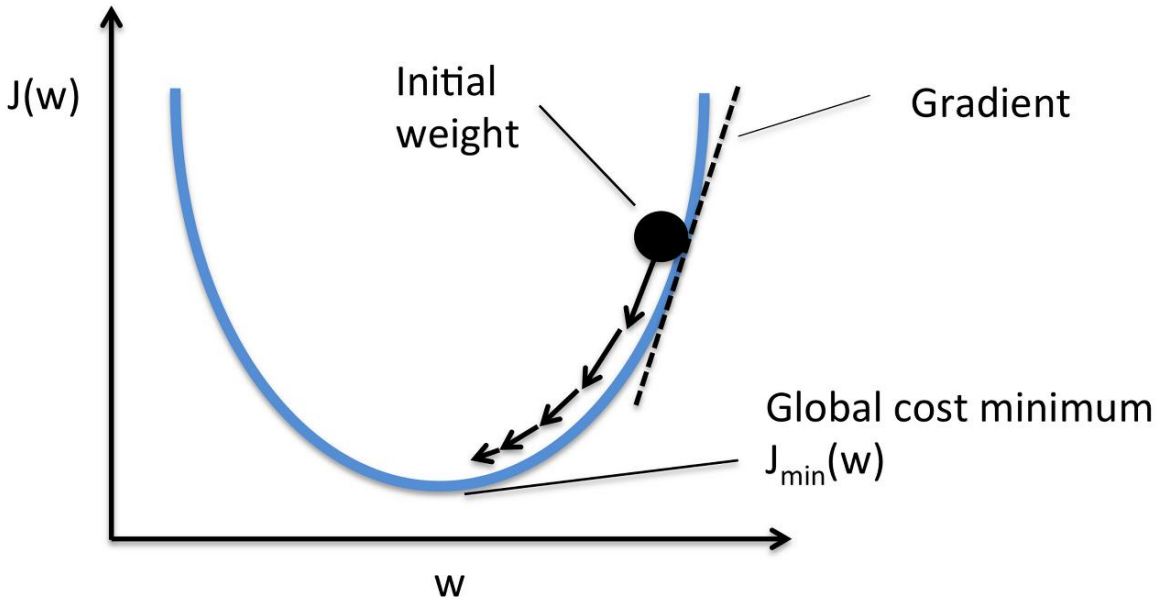
Principle of Model

- How to make less loss?
  - Loss의 정확한 형태를 알 수는 없다.
    - Loss의 gradient를 찾아서 최소인 지점을 찾아가야함.



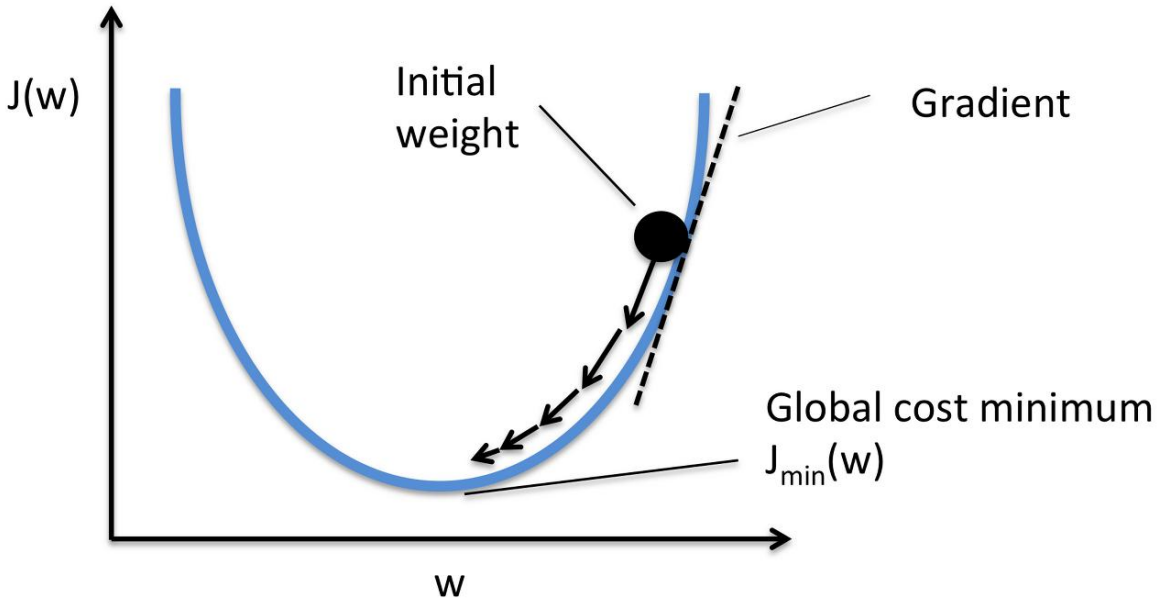
Optimizer

- What optimizer do?
  - Loss가 최소인 지점을 찾아 w를 업데이트
    - 최소인 지점을 찾아가는 방법은 다양함.
    - 다양한 optimizer 존재 (SGD, Adam..)



Optimizer

- Gradient descent algorithm
  - Gradient를 구하여 learning rate(η)에 따라 이동하며 최소 loss를 찾는다.
    - 최소 loss에 다가감에 따라 gradient 감소
      - W에 gradient x lr를 빼주면서 업데이트

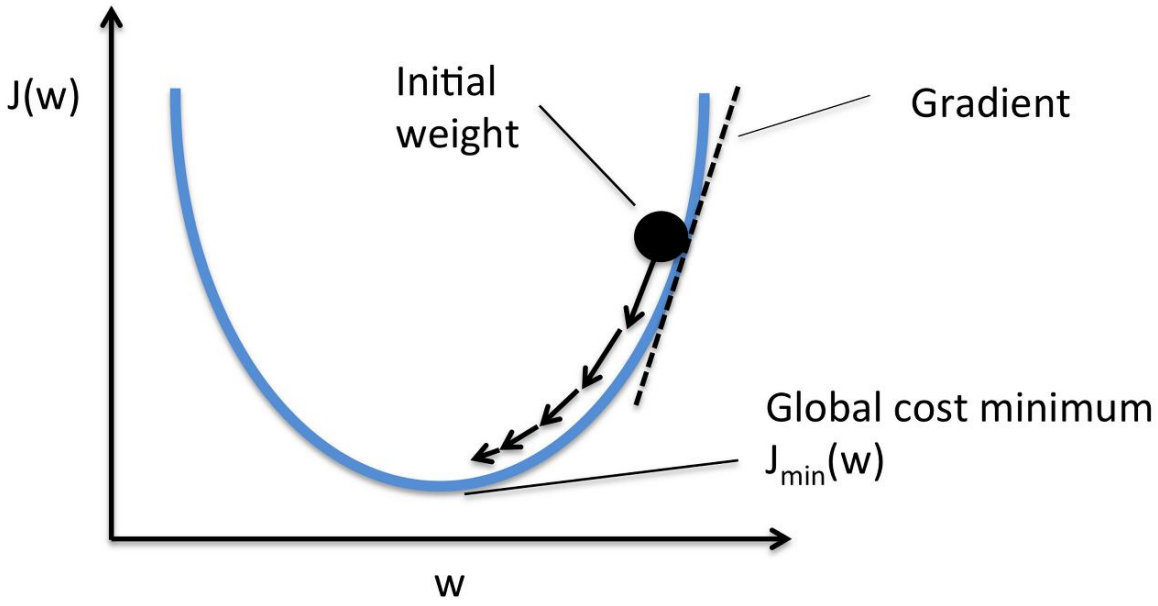


$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

Gradient descent

Optimizer

- Gradient descent algorithm
  - Gradient를 구하여 learning rate(η)에 따라 이동하며 최소 loss를 찾는다.
    - Lr이 너무 작으면 optimization이 너무 느리게 일어남
    - Lr이 너무 크면 최소점을 지나치거나 W가 발산함



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

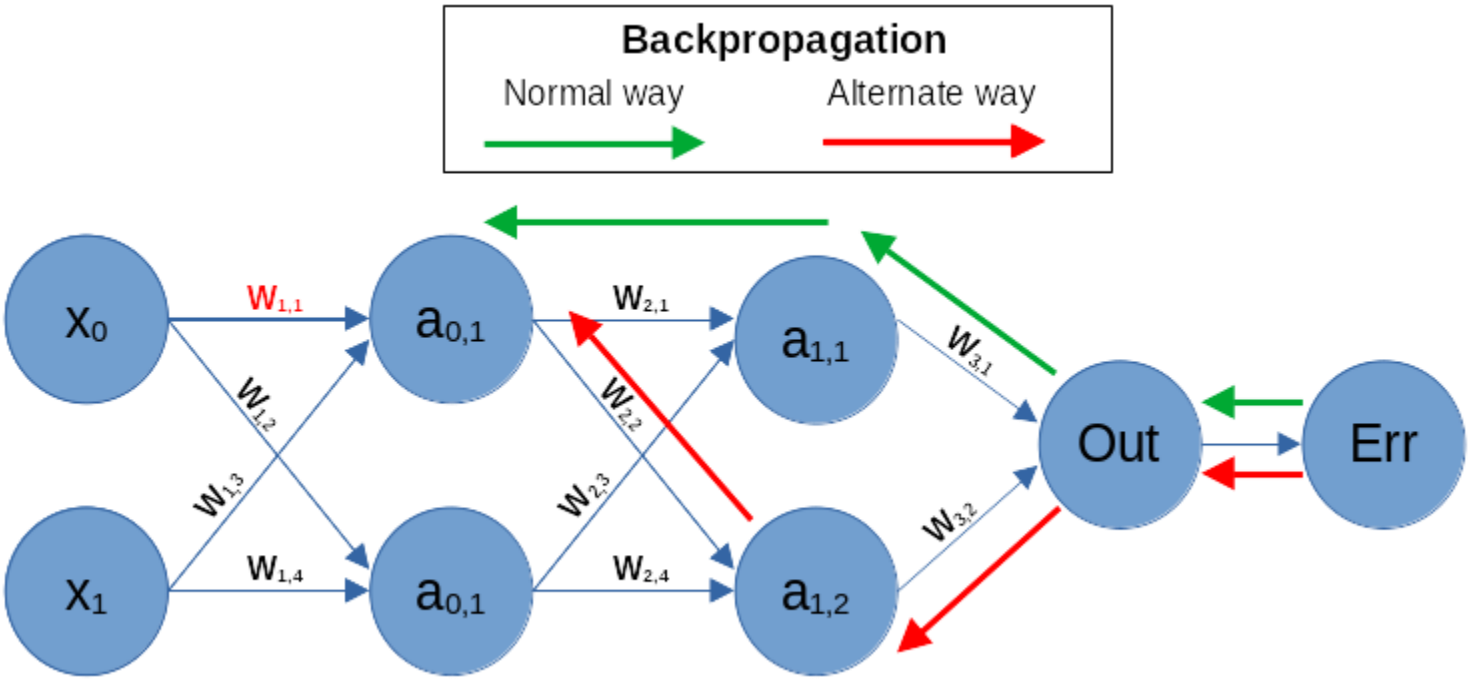
Gradient descent

Optimizer

- Back propagation
  - How to Calculate the Gradient?
    - Forward propagation의 역과정으로 chain rule을 이용하여 gradient를 구함.

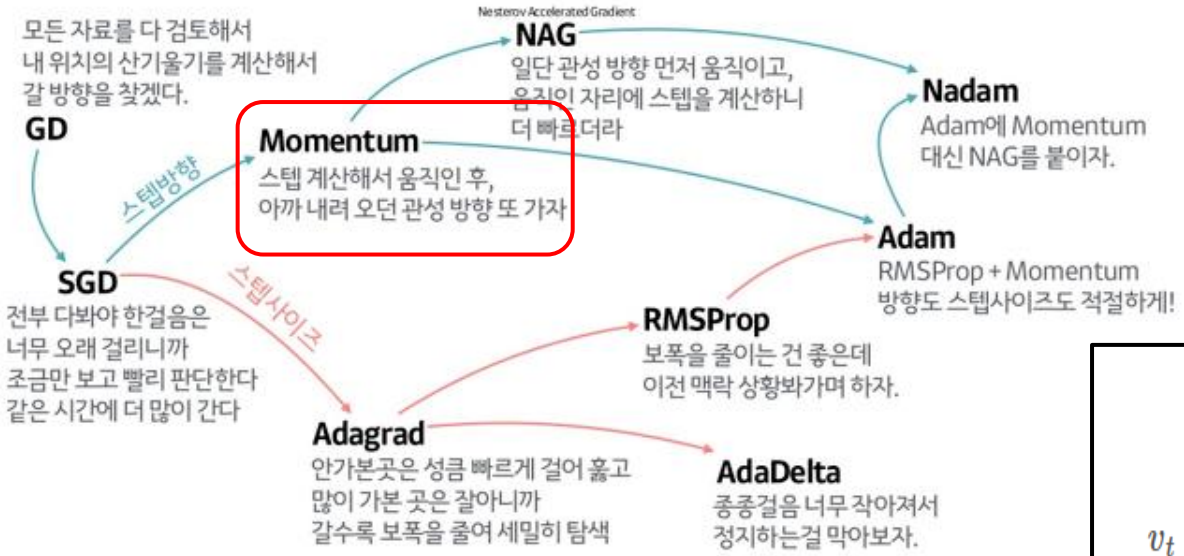
      ∴ y= f(wx+b)인 경우, u=wx+b , L = y* -y $\longrightarrow$ $\frac{\partial L}{\partial w} = (\frac{\partial L}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial w})$

      ✓W에 대한 loss의 **gradient**를 구함



Optimizer

- Another optimizer
  - SGD는 현재 상태의 gradient만을 반영하여 local minima, saddle point에 빠질 수 있음.
    - 이전까지의 gradient를 반영하여 더해주는 momentum 사용
      - Momentum으로 인해 Local minima에서 탈출 가능



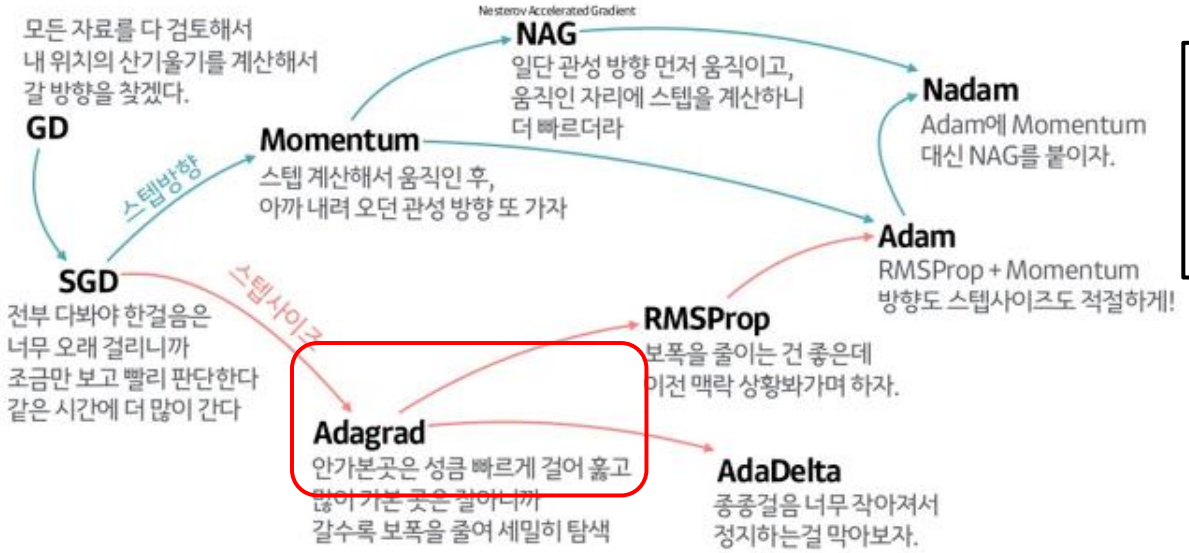
$$\theta = \theta - v_t$$

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$

NAG

Optimizer

- Another optimizer
  - SGD는 각 파라미터의 Gradient를 고려하지 않고 고정된 learning rate를 사용
    - 각 파라미터별로 gradient의 크기와 업데이트 빈도는 서로 다름
      - Ex) gradient가 항상 큰 파라미터는 진동(oscillation) 심함, 최적점 근처에서 불안정
  - Adagrad는 각 파라미터의 과거 gradient 정보를 누적하여 learning rate에 반영
    - 자주 업데이트되는 파라미터는 학습 속도가 줄고, 드물게 업데이트되는 파라미터는 더 크게 학습
      - 누적 gradient가 분모에 위치하기 때문



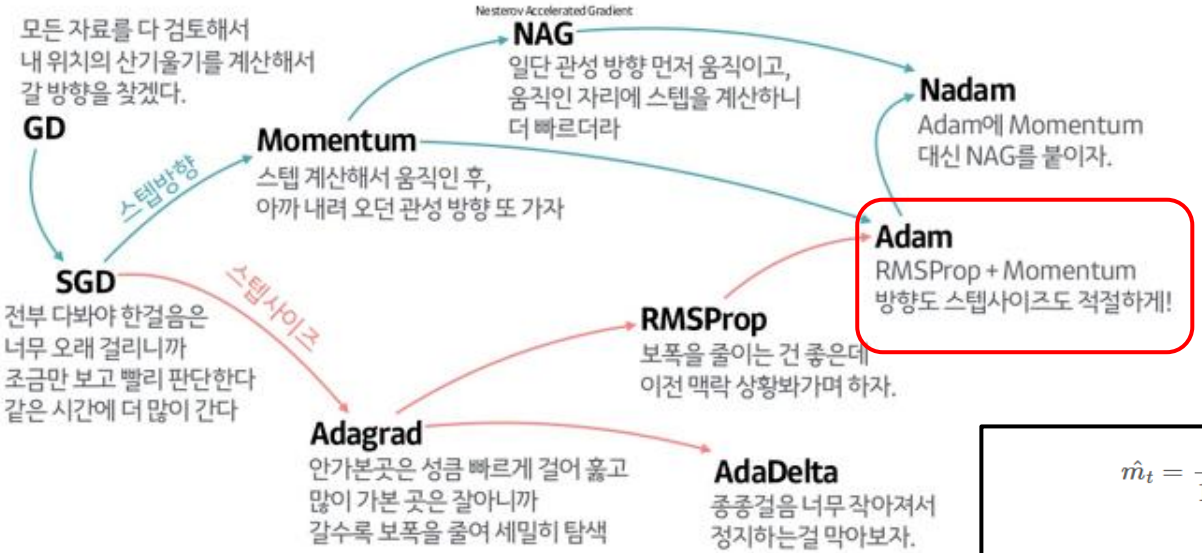
$$\theta_{t+1} = \theta - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot \nabla_\theta J(\theta_t)$$

$$G_t = G_{t-1} + (\nabla_\theta J(\theta_t))^2$$

Adagrad

Optimizer

- Another Optimzier
  - Adam은 momentum과 RMSProp의 장점만을 모아 만든 기법
    - 스텝마다 다른 learning rate + momentum 적용



$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

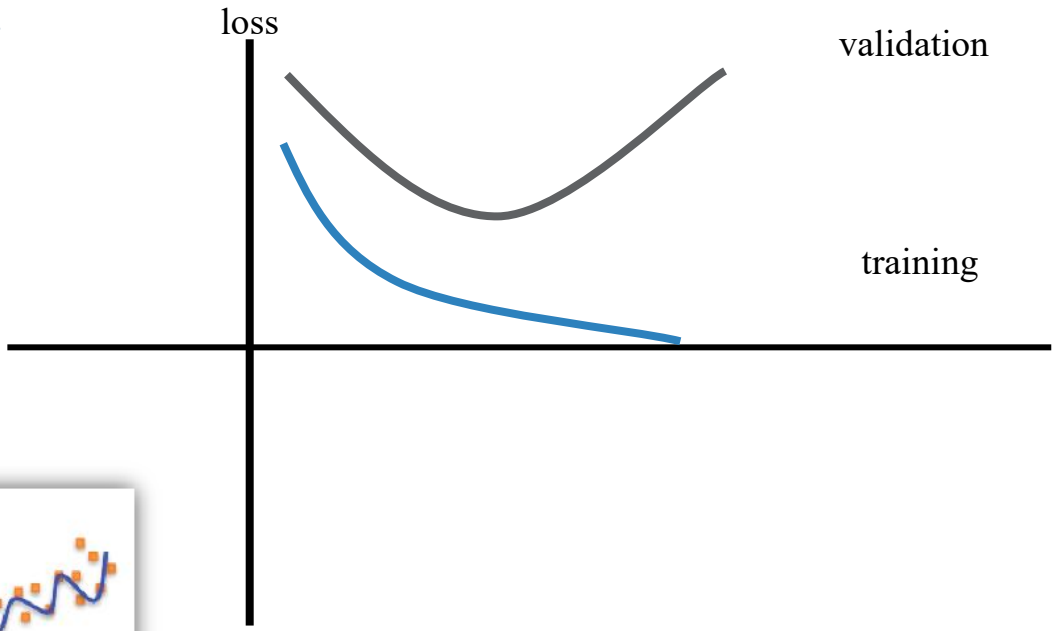
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

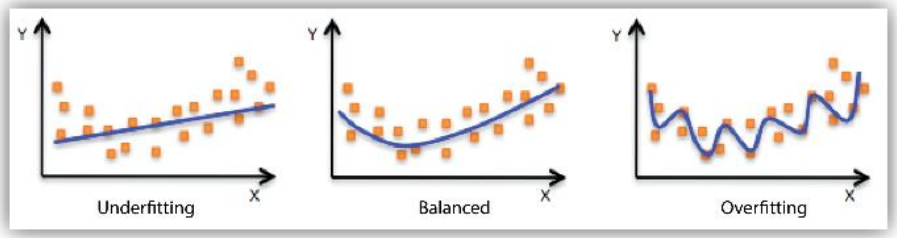
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

Adam

Problems & Technic

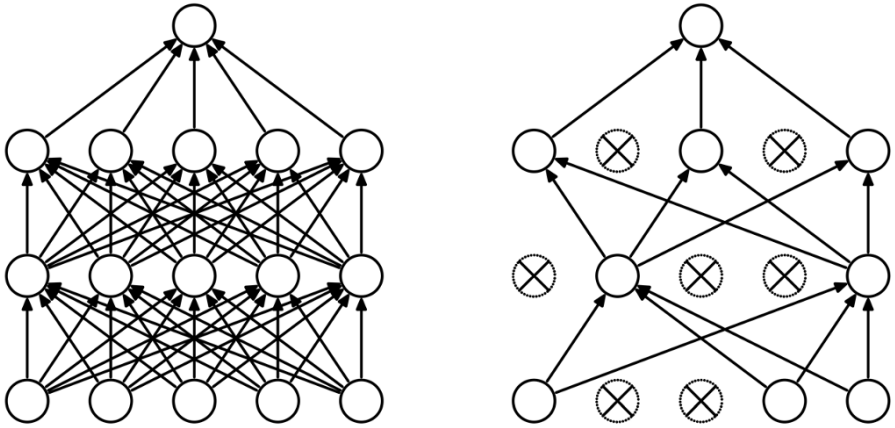
- What is overfitting?
  - 최적화한 모델이 trainset에 과도하게 맞춰져 실제 학습에서는 성능이 떨어짐
    - 학습이 일어날 수록 train loss감소, valid loss 감소 x
    - 과도하게 비선형성을 띄는 모습.
      - ∴ 특정 training set에만 적합





Problems & Technic

- How to solve?
  - 학습 정도를 줄인다.
  - 바이어스를 추가한다.



(a) Standard Neural Net

(b) After applying dropout.

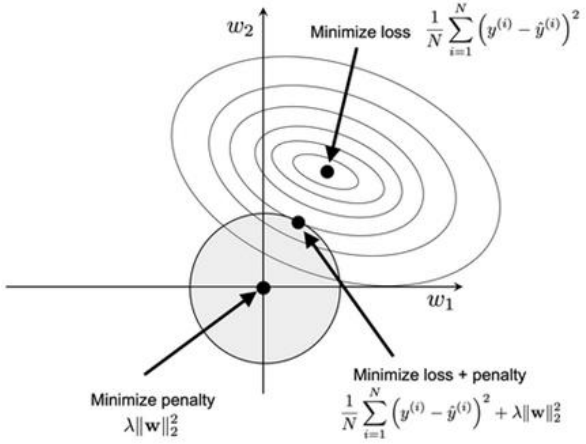
특정 뉴런에 의존하는
학습을 막기위해
무작위로 뉴런을 삭제,

좀더 일반화된 특징을
학습하게 됨

Drop out

Problems & Technic

- How to solve?
  - L2 regularization
    - 너무 큰 가중치(w)를 갖지 못하게 함 (모델이 복잡해지는 것 방지)
      - 가중치 패널티를 부여하여 제한을 둠
      - λ가 커질수록 제한정도 증가.



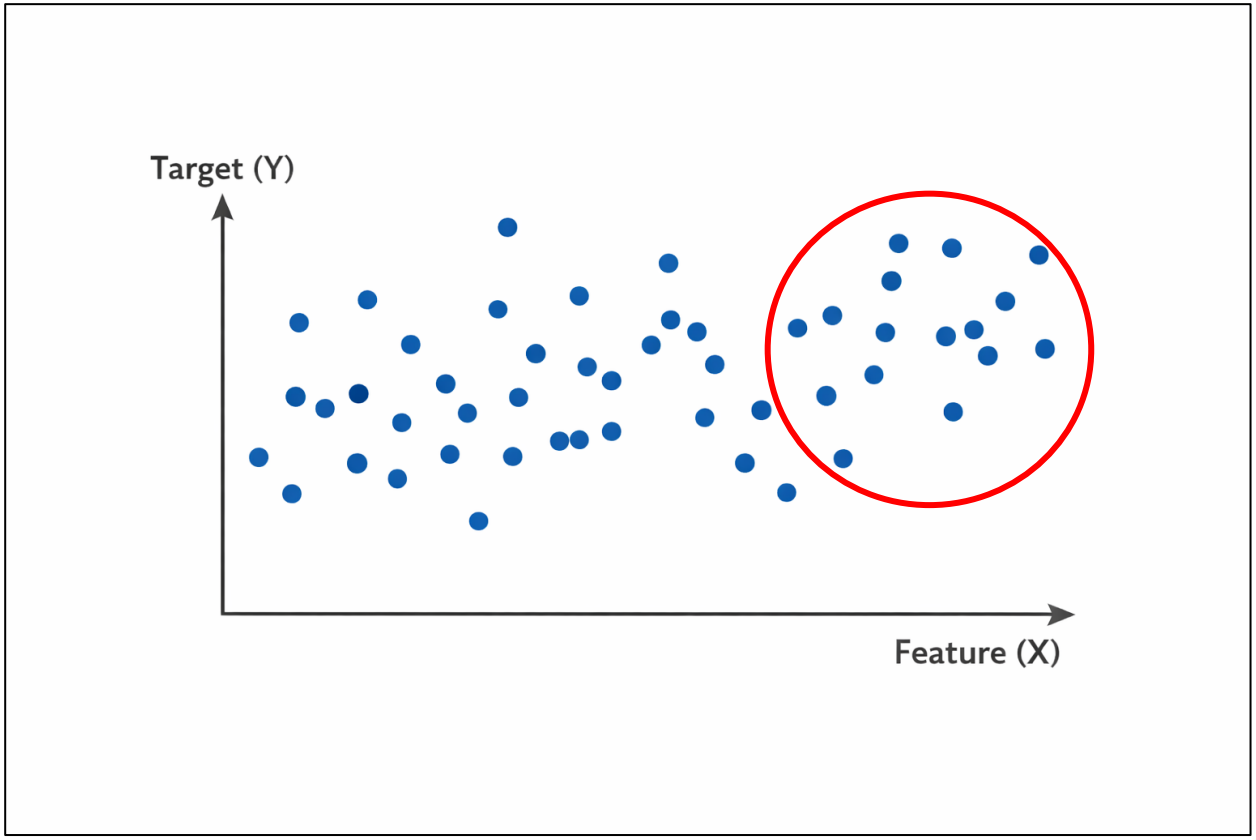
L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

L2 regularization

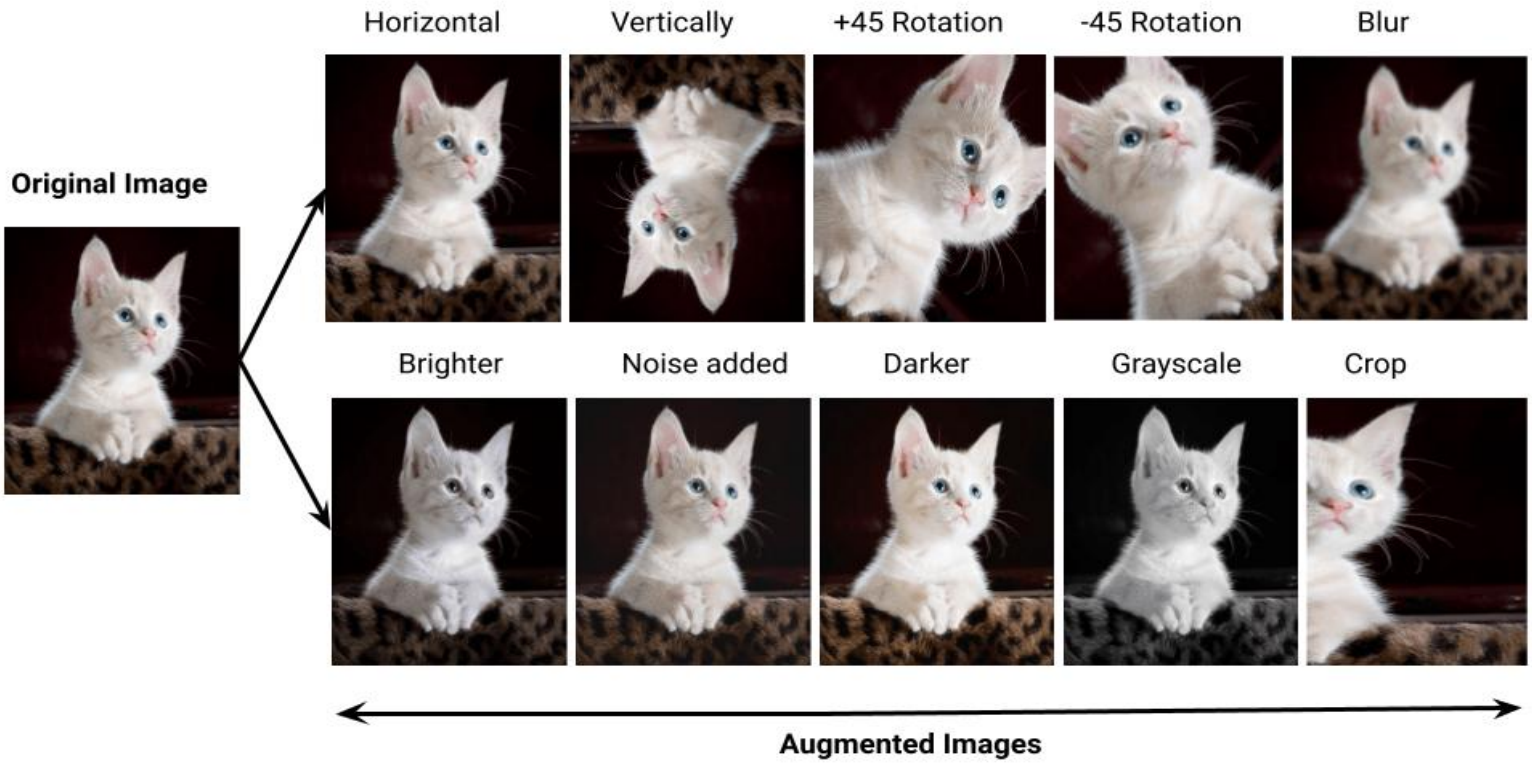
Problems & Technic

- When overfitting happen?
  - Dataset의 문제로 loss자체가 잘못된 경우 발생
    - Ex) 보편적인 data가 아닌 지엽적인 data인 경우



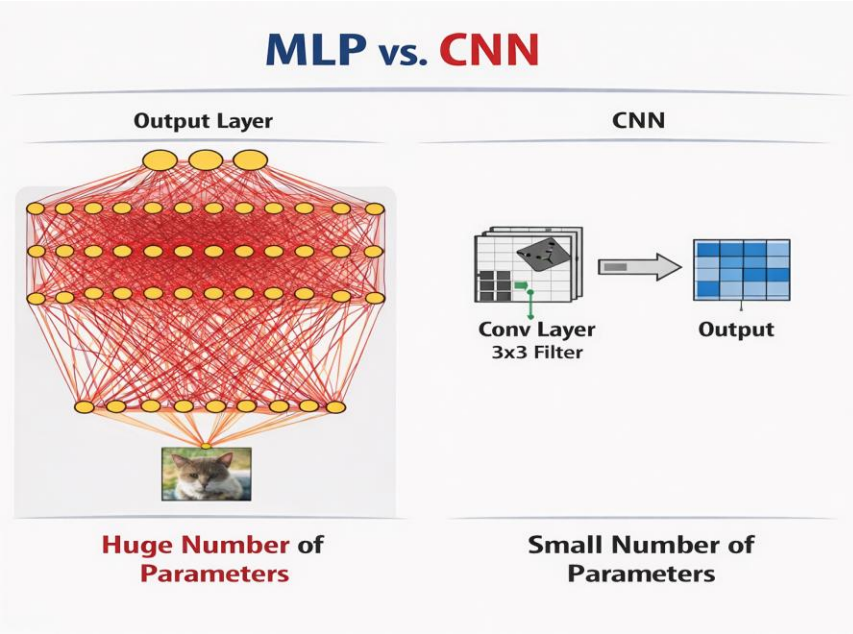
Problems & Technic

- How to solve
  - Data Augmentation
    - Data 학습을 어렵게 하기 위해 변화를 적용
      - 밝기 조절, noise 적용, crop(자르기) 등
    - Model이 학습하는 데이터의 다양성 증가



CNN

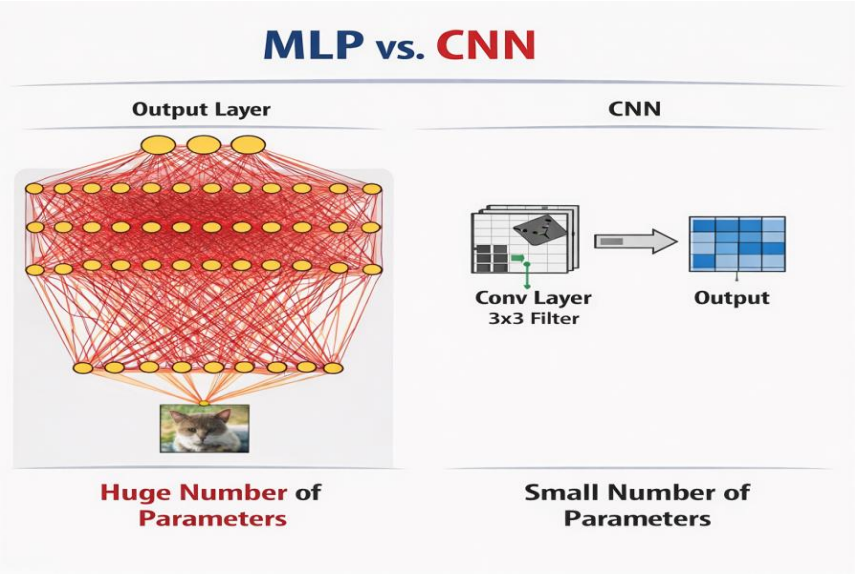
- CNN vs MLP
  - problem of MLP
    - 쓸데없는 배경정보를 포함
      - MLP는 입력과 동시에 입력의 모든 정보가 모델전체와 연결됨
    - 파라미터수의 폭증
      - (32x32x3의 이미지를 MLP에 넣을 때 hid dimension이 512라면 3072x512의 파라미터)
    - 공간적 데이터 무시
      - RGB 이미지는 3차원 데이터인데 이를 1차원으로 변환 후 적용

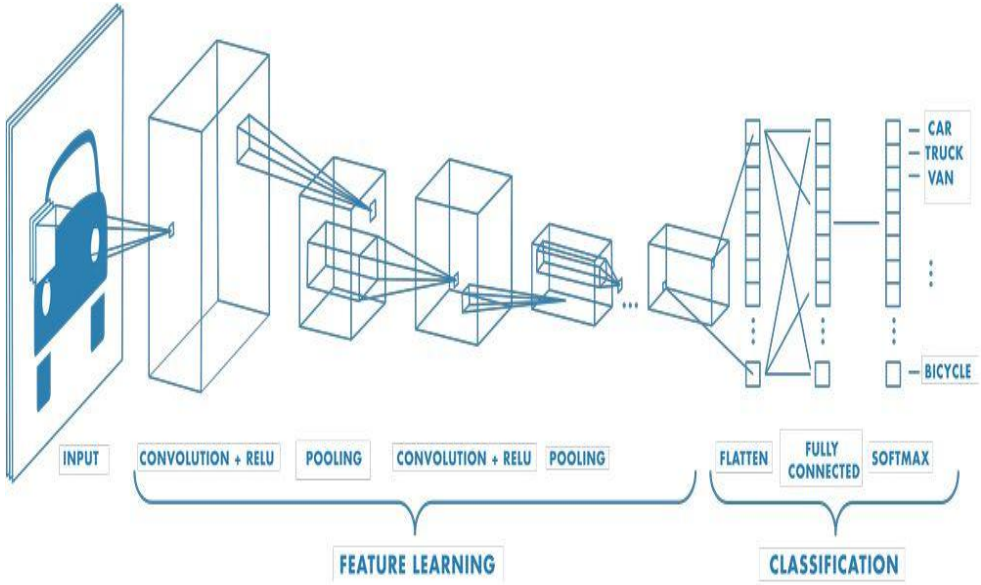




CNN

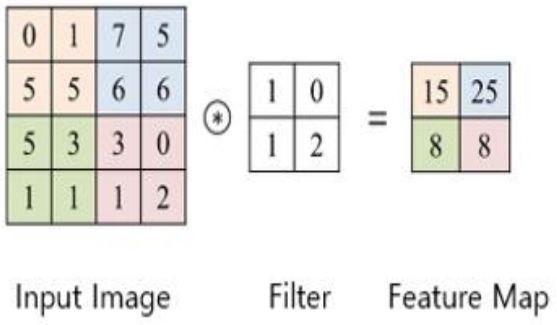
- CNN vs MLP
  - Advantage of CNN
    - CNN은 초반에 conv필터로 이미지의 국소적인 특정 부분만 출력
      - 불필요한 배경정보는 약화되면서 시작됨
    - 파라미터 수 감소
      - 32x32x3이미지에 3x3x3 kernel을 적용 3024x9개의 파라미터
    - 입력의 3차원 데이터 차원을 보존
      - Filter의 출력도 3차원 형태

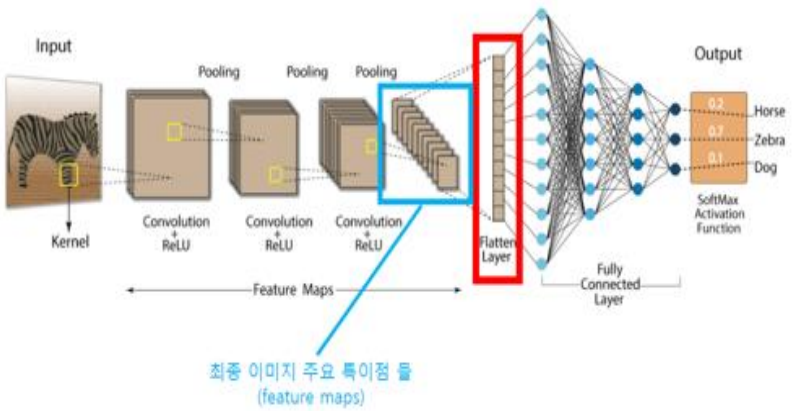




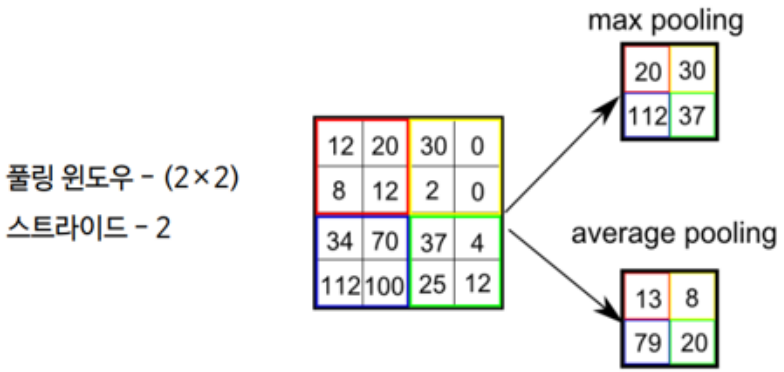
CNN

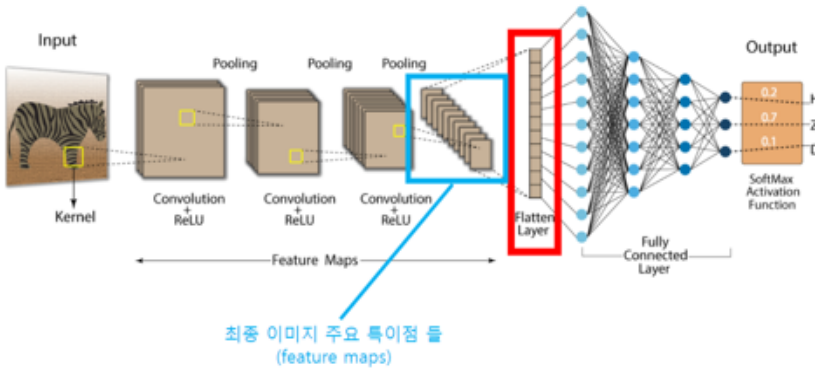
- CNN의 구조
  - conv filter + activation function + pooling
    - 이후 MLP를 통해 최종 출력
  - Filter
    - Filter와 image layer의 오버랩 된 픽셀 값들을 dot product
    - Filter와 유사한 영역이 입력에 얼마나 있는지를 feature map에 나타냄.
    - Feature Map의 값이 클수록 filter의 특징을 많이 포함





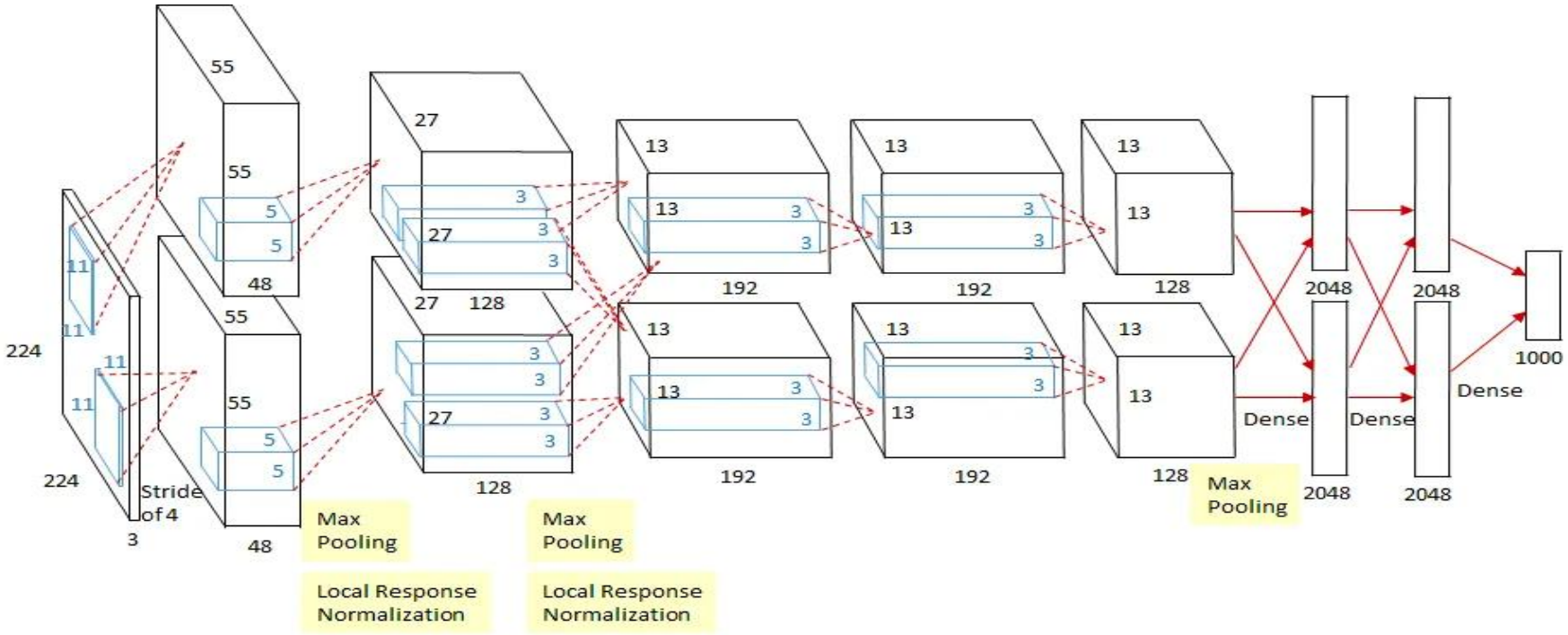
- CNN의 구조
  - conv filter + activation function + pooling
    - 이후 MLP를 통해 최종 출력
  - Pooling
    - Pooling층은 이미지의 크기를 줄여 연산량을 줄여줌.
      - 파라미터의 개수가 줄어 오버피팅을 방지
    - Max pooling은 영역에서 가장 큰 값을 대표값으로 추출
      - 가장 큰 값이 출력에 가장 큰 영향을 줌
    - Average Pooling은 영역의 평균값을 대표값으로 추출
      - Feature의 모든 값을 이용





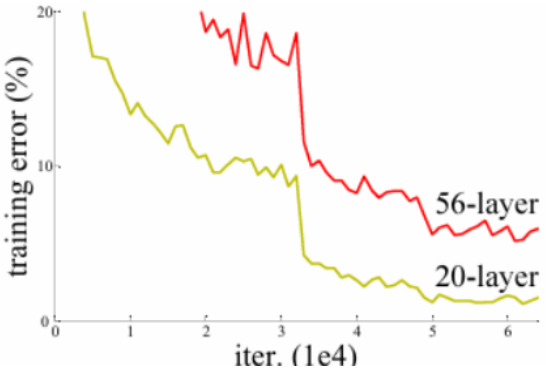
CNN

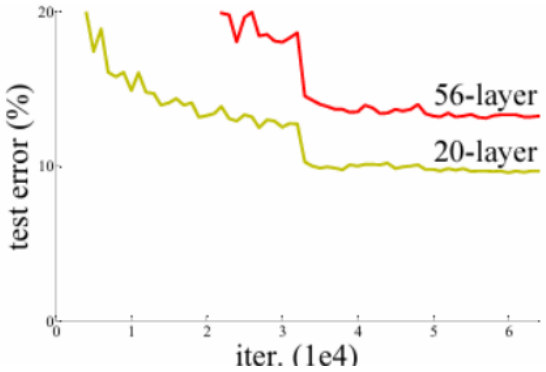
- Another CNN Model
  - AlexNet
    - 최초의 대규모 CNN모델
      - 크기가 큰 filter와 MaxPooling을 사용
    - Relu + dropout을 최초로 사용하여 성능 개선

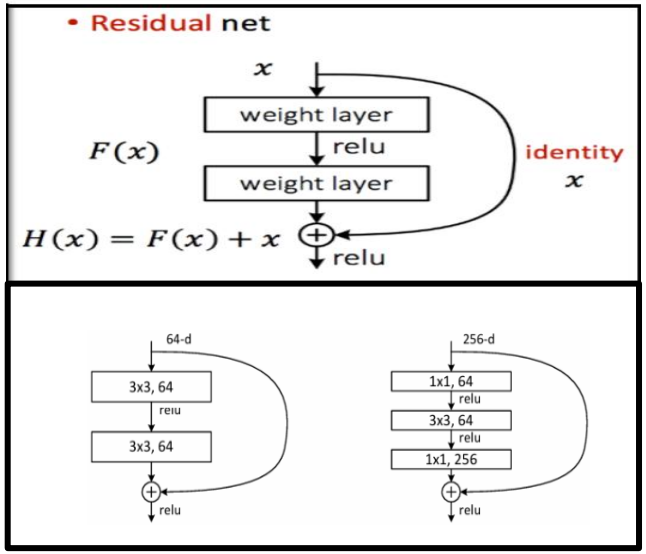


Model

- Another CNN Model
  - Background of ResNet
    - 네트워크의 depth가 깊어질 수록 degradation(성능저하)발생
      - ∴ 깊게 쌓을수록 **gradient vanishing** 심화
    - 깊이 쌓아도 성능이 나빠지지 않는 구조 필요
  - ResNetskip connection을 통해서 결과를 F(x) + x로 만든다(residual block 구조)
    - ∴ H(x)=f(x)+x , f(x) = H(x) – x 학습해야 할 f(x) = H(x) – x
    - back propagation시 gradient가 최소 1이상의 값을 가짐, gradient vanishing 문제 완화
    - 50층이상의 resnet에서는 bottleneck구조의 residual block을 사용



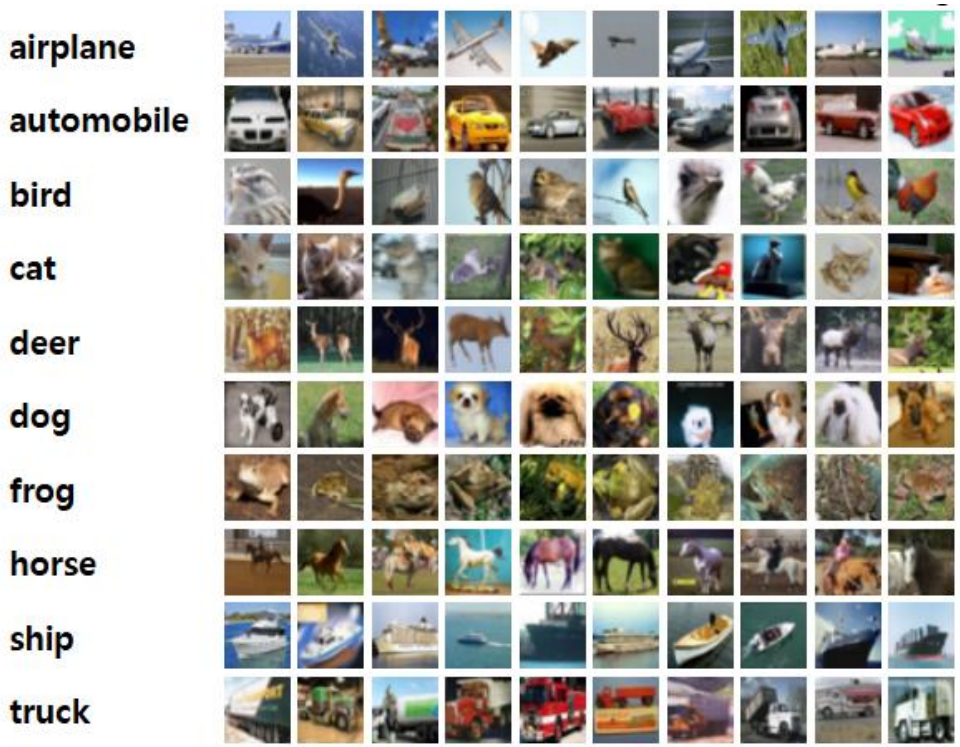




My experiments

- Cifar10 dataset classification
  - Task : Cifar10 dataset image 50000개를 train, validation, testset으로 나누어 학습 후 10개의 class에 분류

Cifar10 dataset



My experiments

- Cifar10 dataset classification
  - 실험 결과 **(MLP)**
    - LR, Model capacity , dropout등 여러가지 hyperparameter tuning 적용

Lr		**Hidden Dim.**	**Drop.**	**#Layers**	**Opt.**	**Ep.**	**Acc.**
Init.	Last.						
0.005	-	1500	0.2	3	Adam	20	54.39
0.0005	-	1500	0.2	3	Adam	50	55.26
0.0005	0.00005	1500	0.2	3	Adam	50	55.68
0.0005	0.00001	2000	0.3	3	Adam	50	56.07
0.0005	**0.00001**	**2000**	**0.2**	**3**	**Adam**	**50**	**56.78**

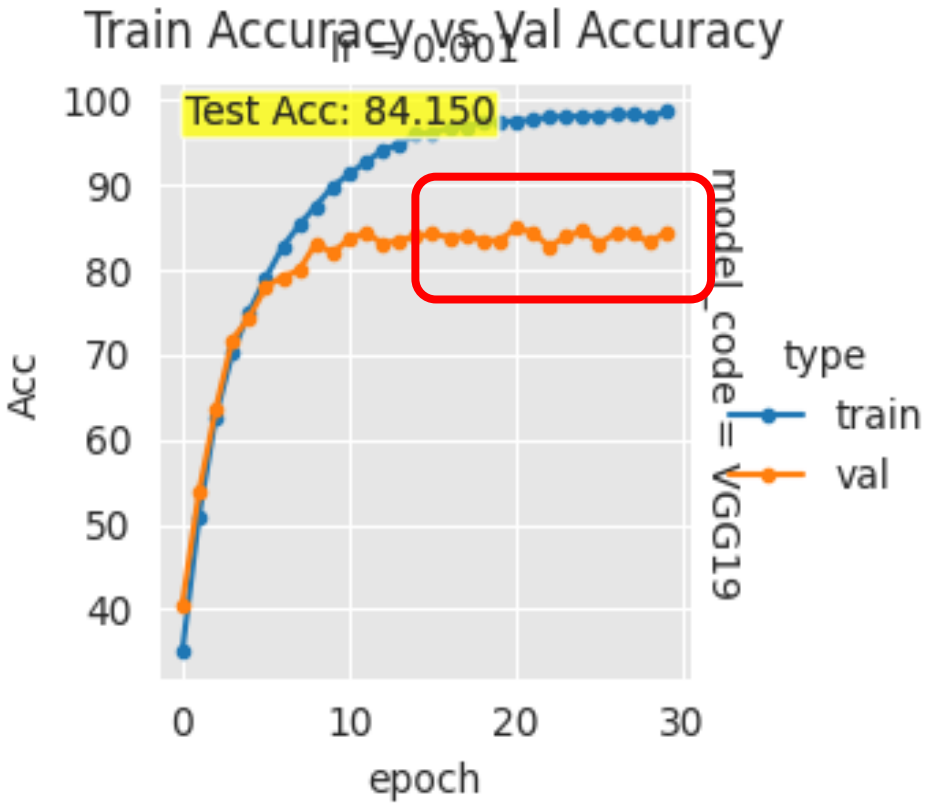
My experiments

- Cifar10 dataset classification
  - 실험 결과 (CNN)
    - MLP 대비 성능 크게 향상
    - Model의 Size가 커질수록 성능 향상

Model (#Layers)	Lr		Drop.	Opt.	Ep.	Acc.
	Init.	Last.				
MLP (Best)	-	-	-	-	-	56.78
VGG11	0.001	Cos	0.4	Adam	20	81.89
VGG13	0.001	Cos	0.4	Adam	20	83.52
VGG16	0.001	Cos	0.4	Adam	40	84.11
VGG19	**0.001**	Cos	**0**	**Adam**	**30**	**84.15**

My experiments

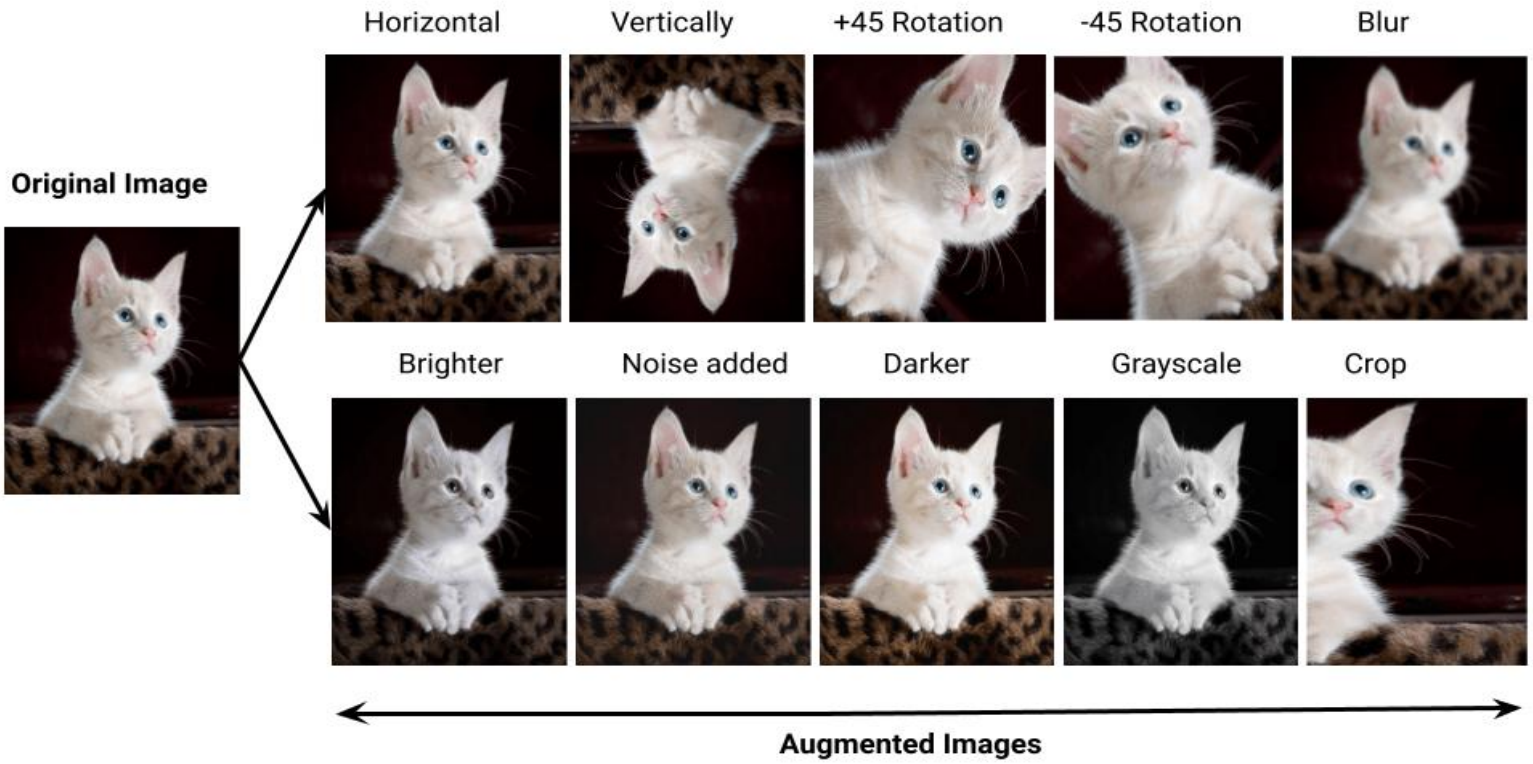
- CNN을 통한 Cifar10 dataset classification
  - 성능 Saturation
    - 여러가지 hyperparameter tuning 적용으로는 성능 향상의 한계 존재
      - Best Acc.: 84.15



- CNN을 통한 Cifar10 dataset classification
  - 성능 향상을 위한 기법
    - (1) Data Augmentation
      - Basic Data Augnmentation
      - CutMix : Regularization Strategy to Train Strong Classifiers with Localizable Features (ICCV 2019 Oral)
    - (2) weight initialization
      - GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training (NeruIPS 2021)

Another technique

- CNN을 통한 Cifar10 dataset classification
  - Data Augmentation 적용
    - 입력 dataset에 대한 변화필요 → augmentation이용
    - 입력 dataset을 뒤집거나 돌리는 등 변화를 줌.
      - 과해지면 성능이 오히려 감소



Another technique

- CNN을 통한 Cifar10 dataset classification
  - Data Augmentation Detail
    - RandAugment: 여러가지 augmentation중 Num_ops만큼 랜덤하게 골라서 설정한 Magnitude만큼의 강도로 적용

```
transform_train: Unknown = transforms.Compose(
    [transforms.RandAugment(num_ops=3,magnitude=6),#무작위로
    transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
     #임의의 영역을 잘라내고 특정 크기로 리사이즈, # transforms.
    #확률 0.5로 좌우반전
```

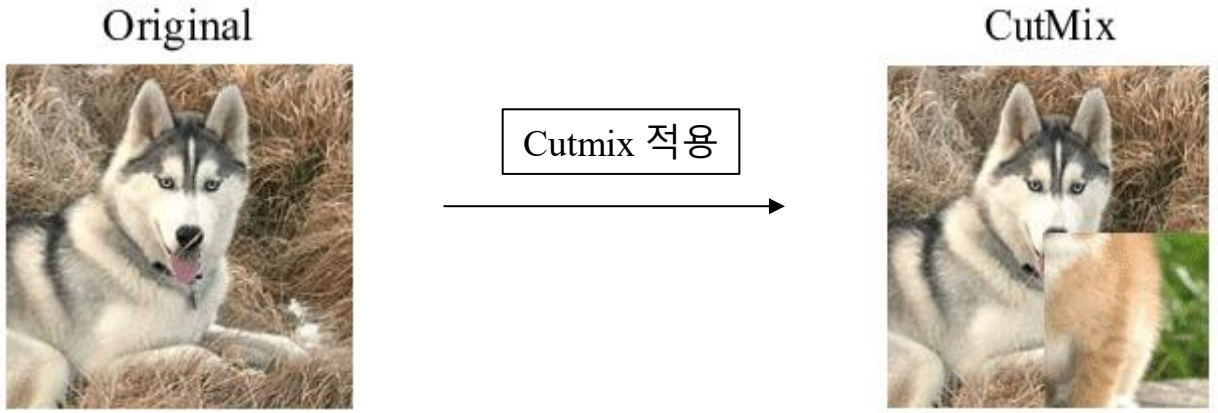
Another technique

- CNN을 통한 Cifar10 dataset classification
  - Data Augmentation 적용
    - 적용 후 모든 모델에서 성능 향상
      - VGG19 모델은 최적화가 더 필요

Model (#Layers)	Default	W/ Augmentation
VGG11	81.89	**88.44**
VGG13	83.52	**90.54**
VGG16	84.11	**91.12**
VGG19	84.15	**90.55**

Another technique

- CNN을 통한 Cifar10 dataset classification
  - Cutmix
    - 서로 다른 label의 이미지의 일부를 섞어 놓음.
      - 정답label 또한 섞은 비율만큼 변화한다.
        - ✓ 늑대 토끼를 cutmix: y= 0.75 늑대 + 0.25 토끼)
        - ✓validation과 test에서 새로운 입력에 대한 성능 향상
      - Augmentation과 함께 적용



Another technique

- CNN을 통한 Cifar10 dataset classification
  - Cutmix Detail
    - Lam: 섞인 이미지의 비율
      - Lam 1: 원본 (늑대)



Lam 0.75

```
inputs: Unknown, labels_a: Unknown, labels_b: Unknown, lam: Unknown=cutmix(x=inputs,y=labels,alpha=1.0)
outputs: Unknown = net(inputs)
criterion: Unknown = nn.CrossEntropyLoss()

loss: Unknown = lam * criterion(outputs, labels_a) + (1 - lam) * criterion(outputs, labels_b)
loss.backward()
optimizer.step()

train_loss += loss.item()
_, predicted: Unknown = torch.max(outputs.data, 1)
total += labels.size(0)
correct += (predicted == labels_a).sum().item()
```

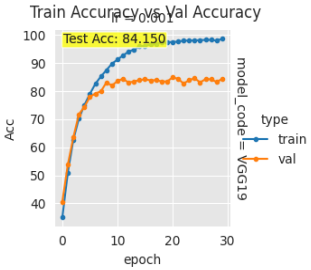
Another technique

- CNN을 통한 Cifar10 dataset classification
  - 실험결과
    - Cutmix 적용 전과 비교해 성능 개선

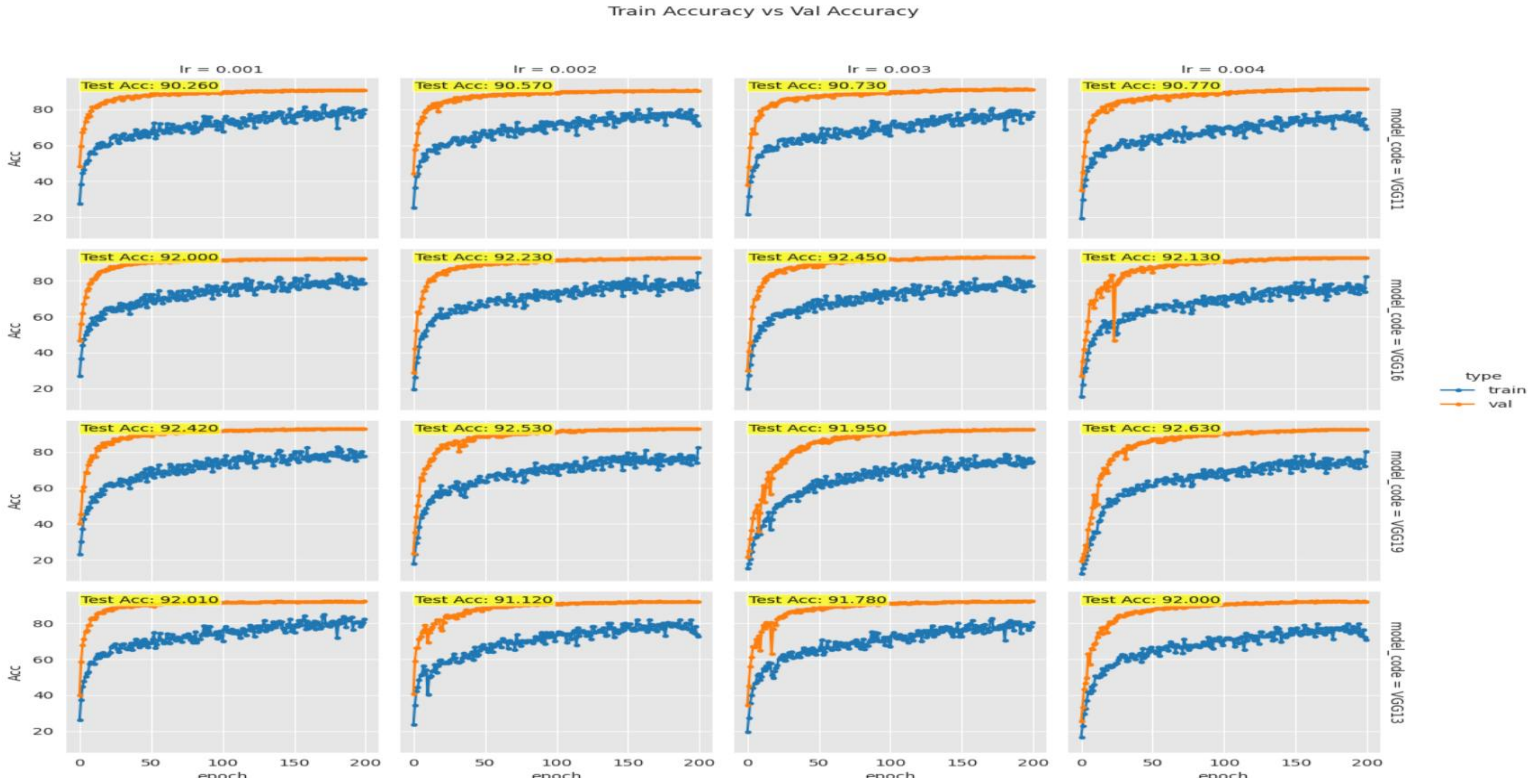
Model (#Layers)	Default	W/ Base Augmentation	W/ Cutmix
VGG11	81.89	88.44	**90.77**
VGG13	83.52	90.54	**92.63**
VGG16	84.11	91.12	**92.45**
VGG19	84.15	90.55	**92.01**

Another technique

- CNN을 통한 Cifar10 dataset classification
  - 실험결과
    - Val_acc가 train_acc보다 높은 정확도를 보임
      - CutMix 적용 이후, Training set이 더 어려운 샘플(hard samples)로 구성되면서 Train Acc는 적용 전보다 낮아짐
      - 대신 데이터 다양성이 크게 증가하여 모델의 일반화 성능이 향상되어 높은 Val 성능을 보여줌



적용 전



Gradinit initialization

- GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training (NeurIPS 2021)
  - 핵심 논리
    - 기존 초기화 방법은 각 layer의 input dimension과 output dimension을 이용해 가중치 분포의 분산을 결정
    - gradinit은 Init 후 분산 초기화 파라미터 s_n를 이용하여 최적의 파라미터로 업데이트
      - s_n 에 대해 optimizing
        - ✓모델의 Loss를 활용하기에 그 모델에 적합한 weight init을 하게 됨
    - => 그레디언트의 분산이 감소하여 안정적인 학습 가능

$$\theta(s) = \{ s_1 W_1^{(0)},\ s_2 W_2^{(0)},\ \ldots,\ s_L W_L^{(0)} \}$$

Xavier initialization

$$W \sim \mathcal{N}\left(0,\ \frac{2}{\text{fan-in} + \text{fan-out}}\right)$$

Gradinit initialization

$$\min_{\eta}\ L(\theta_0 - \eta \nabla L(\theta_0))$$

Gradinit initialization

- GradInit: Learning to Initialize Neural Networks for Stable and Efficient Training (NeurIPS 2021)
  - 실험 방법
    - He initialization으로 초기 weghit Init 후 loss 측정
    - training전 one-step optimize를 시행하여 최소 loss를 만드는 스케일 파라미터 업데이트
      - 초기 스케일 파라미터값 = 1
    - 이때, gradient norm이 특정값(보통 r=1)을 넘지 않아야 하는 제약조건 만족 필요
      - 제약조건 없이 norm이 증가하면 gradient exploding 발생
    - 여러회 반복하여 최적의 스케일 파라미터로 업데이트

He initialization

$$\|\nabla_\theta L(\theta_0)\|_2 \leq \gamma$$

Gradient norm

$$W \sim \mathcal{N}\left(0, \frac{2}{\text{fan-in}}\right)$$

Gradinit initialization

$$\min_\eta \; L(\theta_0 - \eta \nabla L(\theta_0))$$

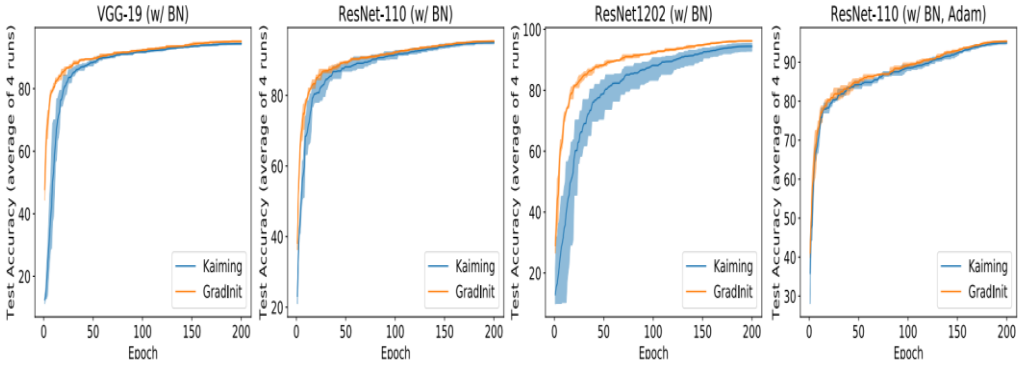
$$\theta(s) = \{ s_1 W_1^{(0)}, \; s_2 W_2^{(0)}, \; \ldots, \; s_L W_L^{(0)} \}$$

$$W_l = s_l \cdot W_l^{(0)}$$

Gradinit initialization

- CNN을 통한 Cifar10 dataset classification
  - 실험 결과
    - kaiming initialization과 비교하여 성능증가.
    - gradinit에서의 gradient 분산이 상대적으로 작음.
    - gradient 분산의 감소가 성능 증가의 원인

Model (# Params)		VGG-19 w/o BN (20.03M)	VGG-19 w/ BN (20.04M)	ResNet-110 w/o BN (1.72M)	ResNet-110 w/ BN (1.73M)
Kaiming	Acc_1	29.1 ± 1.5	12.6 ± 0.6	16.1 ± 2.1	23.2 ± 0.9
	Acc_{best}	94.5 ± 0.1	94.4 ± 0.1	94.2 ± 0.1	95.0 ± 0.2
+1 epoch (Const. LR)	Acc_1	37.2 ± 1.1	19.6 ± 4.0	21.0 ± 3.8	32.5 ± 3.8
	Acc_{best}	94.4 ± 0.1	94.5 ± 0.1	93.9 ± 0.4	94.7 ± 0.3
+1 epoch (Warmup)	Acc_1	37.4 ± 1.2	53.5 ± 2.9	19.8 ± 0.5	48.7 ± 1.1
	Acc_{best}	94.4 ± 0.1	94.7 ± 0.1	94.1 ± 0.1	95.1 ± 0.1
MetaInit	Acc_1	30.5 ± 0.9	35.1 ± 0.6	14.6 ± 2.2	29.0 ± 1.5
	Acc_{best}	94.6 ± 0.1	94.6 ± 0.1	94.2 ± 0.1	94.8 ± 0.1
GradInit	Acc_1	29.3 ± 0.6	47.8 ± 1.8	36.2 ± 0.8	38.2 ± 0.9
	Acc_{best}	$\mathbf{94.7} \pm 0.1$	$\mathbf{95.1} \pm 0.1$	$\mathbf{94.6} \pm 0.1$	$\mathbf{95.4} \pm 0.1$



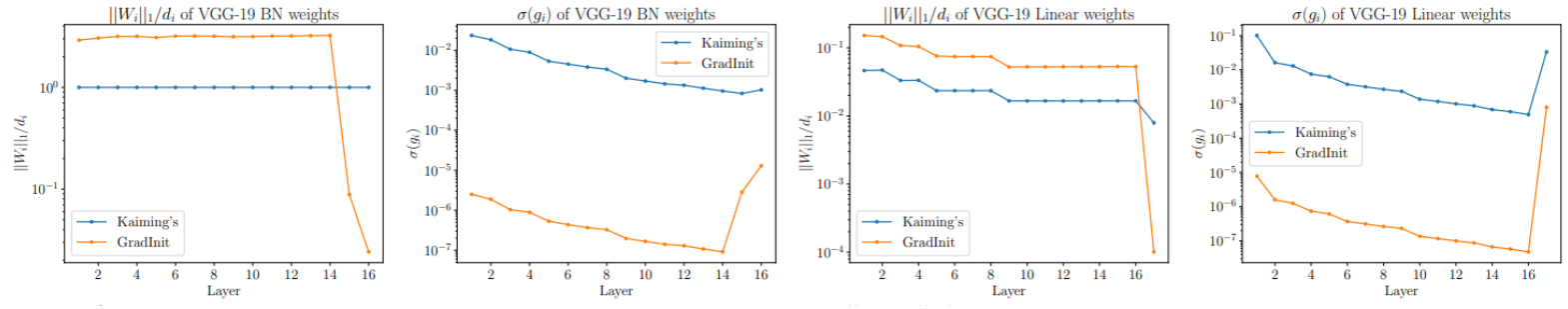
Gradinit initialization

- CNN을 통한 Cifar10 dataset classification
  - gradient 분산 감소의 원인
    - scale parameter α가 학습되며 앞쪽의 conv layer들을 scale up
    - 뒤쪽의 bn_layer 2개와 fc_layer scale down
    - Magnification Effect of BN 감소
      - Bn층이 없을때는 명확한 분산 감소 원인 제시x

$$\frac{\partial L}{\partial(\alpha \boldsymbol{x}_i)} = \frac{\gamma}{n\sqrt{\alpha^2\boldsymbol{\sigma}^2+\epsilon}}\left[n\frac{\partial L}{\partial \boldsymbol{y}_i} - \sum_{j=1}^{n}\frac{\partial L}{\partial \boldsymbol{y}_j} - \frac{\boldsymbol{y}_i-\boldsymbol{\beta}}{\gamma}\sum_{j=1}^{n}\frac{\partial L}{\partial \boldsymbol{y}_j}\cdot\frac{\boldsymbol{y}_j-\boldsymbol{\beta}}{\gamma}\right],$$

$$\frac{\gamma}{\sqrt{\alpha^2\sigma^2+\epsilon}}$$

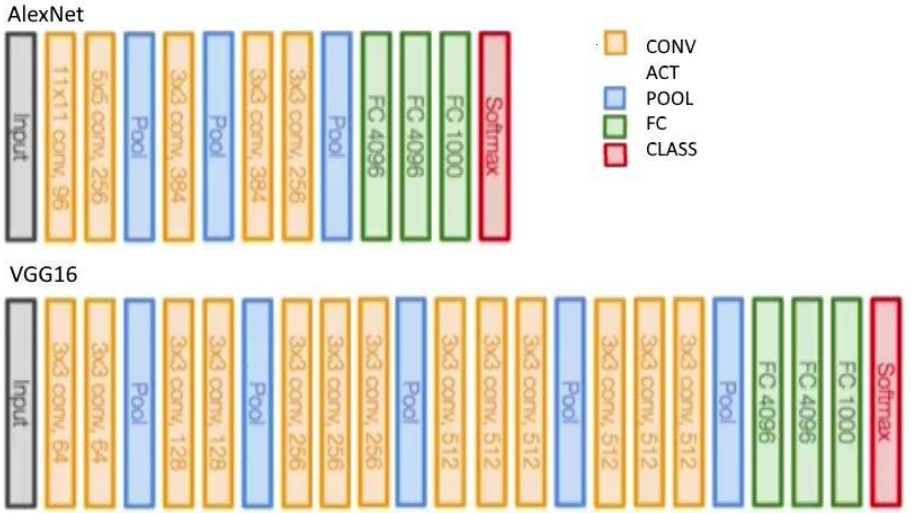
Bn을 기준으로 출력과 입력의 gradient관계



감사합니다

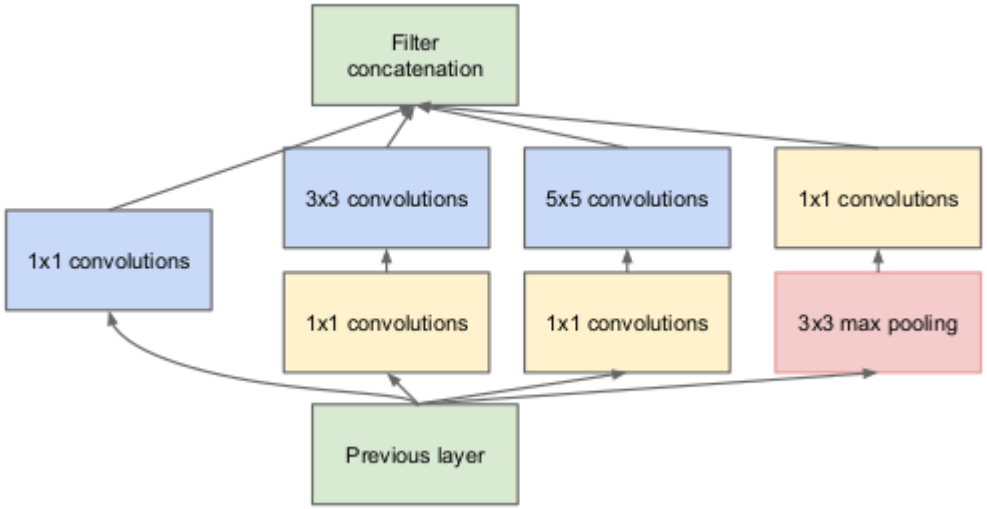
Appendix

- Another CNN Model
  - VGGNet
    - Filter의 크기 감소, layer의 수 증가
      - alexnet 8layers → VGGNet 16~19layers
    - 더 적은 parameter로 더 넓은 receptive field를 가짐.
      - 8x8 image에 3x3 filter3개 적용, 3x3 → 5x5 → 7x7 receptive field증가
    - 비선형성 증가
      - Layer가 깊어질 수록 많은 activation function을 거침
    - Fc(fully connected)층에서 **parameter수 폭증**(overfitting 가능성 증가)

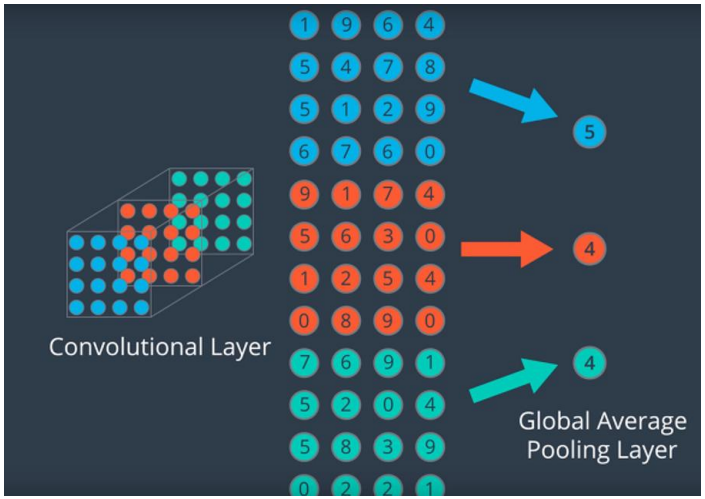


Appendix

- Another CNN Model
  - GoogleNet
    - parameter수를 줄이기 위해 fc layer를 삭제
      - **global average pooling**으로 이를 대체
      - 전체 feature map을 1차원 벡터로 나타냄
    - inception 모듈사용
      - 5x5 3x3 1x1등 여러 스케일을 동시에 보는 효과
      - conv1x1 filter를 통해 파라미터 수 폭증을 억제(conv1x1은 출력의 depth를 줄임.)
        - ex) 32×32×128 and 32×32×32(filter) ⟶ 채널 수 32로 감소

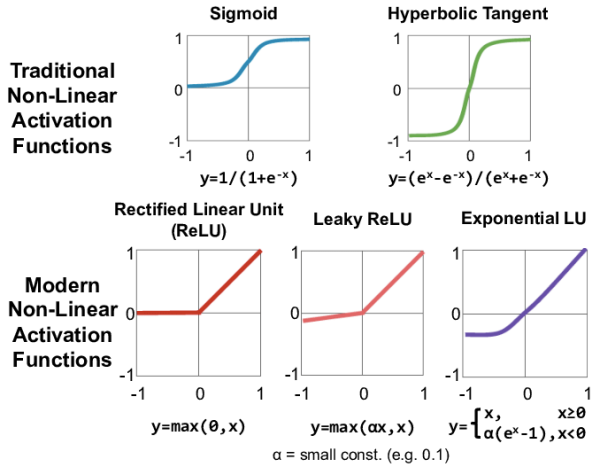


Global pooling



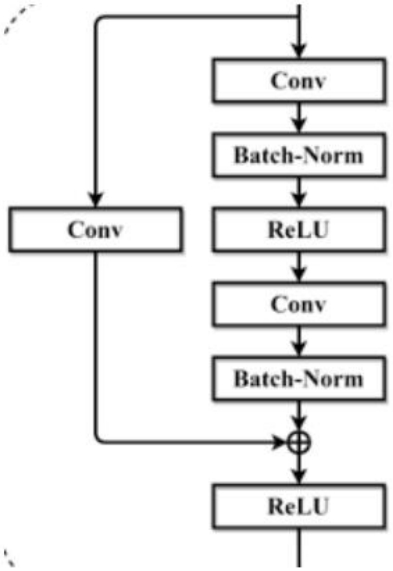
Appendix

- What is gradient vanishing?
  - Backpropagtion 진행중에 gradient가 작아져 입력쪽의 gradient가 0에 가까워짐
    - Optimization 불가
  - Why it happen?
    - Activation 함수의 문제
      - Sigmoid, hyper tangent는 입력이 증가할수록 gradient가 0에 수렴
      - Relu로 이들을 대체하여 사용(기울기 1)
    - Model의 network가 깊어짐



Appendix

- How to solve
  - Batch normalization
    - 입력데이터의 분포를 조정하여 gradient vanishing 방지
      - Ex) relu function 적용전 평균을 0으로 맞추어 입력값이 한쪽을 치우침을 방지.



Batchnormalization layer 순서

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

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

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

batchnormalization

Appendix

- MLP를 통한 Cifar10 dataset classification
  - 실험결과
    - epoch을 증가시켰을때 train_acc향상.
    - dropout 을 증가시켜도 val_acc변화x

	epoch	Hid_dim	dropout	N_layer	optimizer	lr	Train/ val_acc
#cifar10	20	300	0.1	3	Adam	0.0015	67/50
	25	300	0.1	3	Adam	0.0015	75/50
	25	300	0.2	3	Adam	0.0015	70/50
	20	500	0.3	5	Adam	0.005	50/40

Appendix

- MLP를 통한 Cifar10 dataset classification
  - 실험결과
    - hid_dim을 1500이상으로 증가
    - lr 을 학습도중에 낮추어 더 낮은 loss를 찾음
      - 0.005 ⟶ 0.00001 (0.005로 전체 epoch 시행후 test_acc가 일정해지는 구간에서 변화)

	lr	Hid_dim	dropout	N_layer	optimizer	epoch	test_acc
#cifar10	0.005	1500	0.2	3	Adam	20	54.39
	0.0005	1500	0.2	3	Adam	50	55.26
	0.0005 ->0.00005	1500	0.2	3	Adam	50	55.68
	0.0005 ->0.00001	2000	0.3	3	Adam	50	56.07
	0.0005 ->0.00001	2000	0.2	3	Adam	50	56.78

Appendix

- CNN을 통한 Cifar10 dataset classification
  - 각 layer별 실험결과 정리(VGG11, VGG13)

	lr	dropout	optimizer	epoch	test_acc
VGG11	0.0001	0.2	Adam	20	72.64
	0.00001	0	Adam	20	65.69
	0.001->0.00005	0.4	Adam	20	81.62
	0.001	0.4	Adam	20	81.89

	lr	dropout	optimizer	epoch	test_acc
VGG13	0.0001	0.2	Adam	20	70.64
	0.00001	0	Adam	20	63.89
	0.001->0.00005	0.4	Adam	20	83.52
	0.001	0.4	Adam	20	83.43

- lr가 낮았을 때 성능이 나오지 않음
- lr를 0.001로 설정한 후 성능을 높일 수 있었음.

Appendix

- CNN을 통한 Cifar10 dataset classification
  - 각 layer별 실험결과 정리(VGG11, VGG13)

	lr	dropout	optimizer	epoch	test_acc
VGG16	0.001	0.4	Adam	40	84.11
	0.001	0	Adam	30	84.09
	0.001	0.2	Adam	30	83.56
	0.002	0.4	Adam	30	83.73
	0.003	0.4	Adam	30	84.06

	lr	dropout	optimizer	epoch	test_acc
VGG19	0.001	0.4	Adam	40	83.11
	0.001	0	Adam	30	84.15
	0.001	0.2	Adam	30	83.32
	0.002	0.4	Adam	30	83.43
	0.003	0.4	Adam	30	82.39

  - lr을 0.001에서 증가시킴
  - Dropout rate 변화
  - 미세한 차이는 있지만 큰 변화 x

    ⟶ 단순한 hyperparameter tuning으로는 한계

Another technique

- CNN을 통한 Cifar10 dataset classification
  - Data Augmentation 적용
    - hyperparameter tuning만으로는 한계 발생
    - 입력 dataset에 대한 변화필요 ⟶ augmentation이용
    - 입력 dataset을 뒤집거나 돌리는 등 변화를 줌.
      - 과해지면 성능이 오히려 감소

	lr	dropout	Opt.	epoch	test_acc
VGG11	0.001	0	Adam	200	87.59
	0.002	0	Adam	200	88.44
	0.003	0	Adam	200	88.19
	0.004	0	Adam	200	88.5
		0	Adam	200	

	lr	dropout	Opt.	epoch	test_acc
VGG16	0.001	0	Adam	200	90.18
	0.002	0	Adam	200	90.83
	0.003	0	Adam	200	90.41
	0.004	0	Adam	200	91.12
		0	Adam	200	

	lr	dropout	Opt.	epoch	test_acc
VGG13	0.001	0	Adam	200	89.36
	0.002	0	Adam	200	90.27
	0.003	0	Adam	200	90.54
	0.004	0	Adam	200	90.15
		0	Adam	200	

	lr	dropout	Opt.	epoch	test_acc
VGG19	0.001	0	Adam	200	90.49
	0.002	0	Adam	200	90.41
	0.003	0	Adam	200	90.52
	0.004	0	Adam	200	90.55
		0	Adam	200	

Another technique

- CNN을 통한 Cifar10 dataset classification
  - 실험결과
    - cutmix 적용 전과 비교해 성능 개선
    - **vgg11**에서 성능이 떨어짐을 관찰

	lr	dropout	optimizer	epoch	test_acc
VGG11	0.001	0	Adam	200	90.26
	0.002	0	Adam	200	90.57
	0.003	0	Adam	200	90.73
	0.004	0	Adam	200	90.77
		0	Adam	200	

	lr	dropout	optimizer	epoch	test_acc
VGG16	0.001	0	Adam	200	92.00
	0.002	0	Adam	200	92.23
	0.003	0	Adam	200	92.45
	0.004	0	Adam	200	92.13
		0	Adam	200	

	lr	dropout	optimizer	epoch	test_acc
VGG13	0.001	0	Adam	200	92.42
	0.002	0	Adam	200	92.53
	0.003	0	Adam	200	91.95
	0.004	0	Adam	200	92.63
		0	Adam	200	

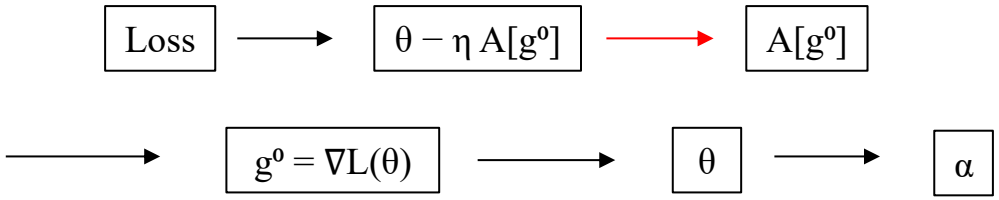
	lr	dropout	optimizer	epoch	test_acc
VGG19	0.001	0	Adam	200	92.01
	0.002	0	Adam	200	91.12
	0.003	0	Adam	200	91.78
	0.004	0	Adam	200	92.00
		0	Adam	200	

Gradinit initialization

- 1-step만 시행하는 이유?
  - 1-step이후는 α를 학습시키는 데 쓸 gradient 구하는 것에 문제 발생

$$\underset{m}{\text{minimize}} \quad L(\tilde{S}; \boldsymbol{\theta}_m - \eta \mathcal{A}[\boldsymbol{g}_{S,\boldsymbol{\theta}_m}])$$

goal



<α에대한 loss gradient 구하는 backpropagation 과정>

$$\mathcal{J}(\alpha) = L\left(\theta^{(1)}\right), \quad \theta^{(1)} = \theta^{(0)} - \eta A[g^{(0)}], \quad g^{(0)} = \nabla_\theta L(\theta^{(0)})$$

$$\frac{\partial \theta^{(1)}}{\partial \alpha} = \frac{\partial \theta^{(0)}}{\partial \alpha} - \eta \underbrace{\frac{\partial A[g^{(0)}]}{\partial g^{(0)}}}_{\text{Jacobian}} \cdot \underbrace{\frac{\partial g^{(0)}}{\partial \theta^{(0)}}}_{\text{Hessian}} \cdot \frac{\partial \theta^{(0)}}{\partial \alpha}$$

Jacobian과 hessian으로 인해 계산량 폭증 문제 발생 → To solve this problem → A[gS,θm]를 상수 취급