

# 딥러닝팀

## 1팀

김예찬

윤지영

채소연

한지원

홍지우

# INDEX

---

1. CNN
2. CNN의 발전 과정
3. RNN
4. RNN 모델의 응용

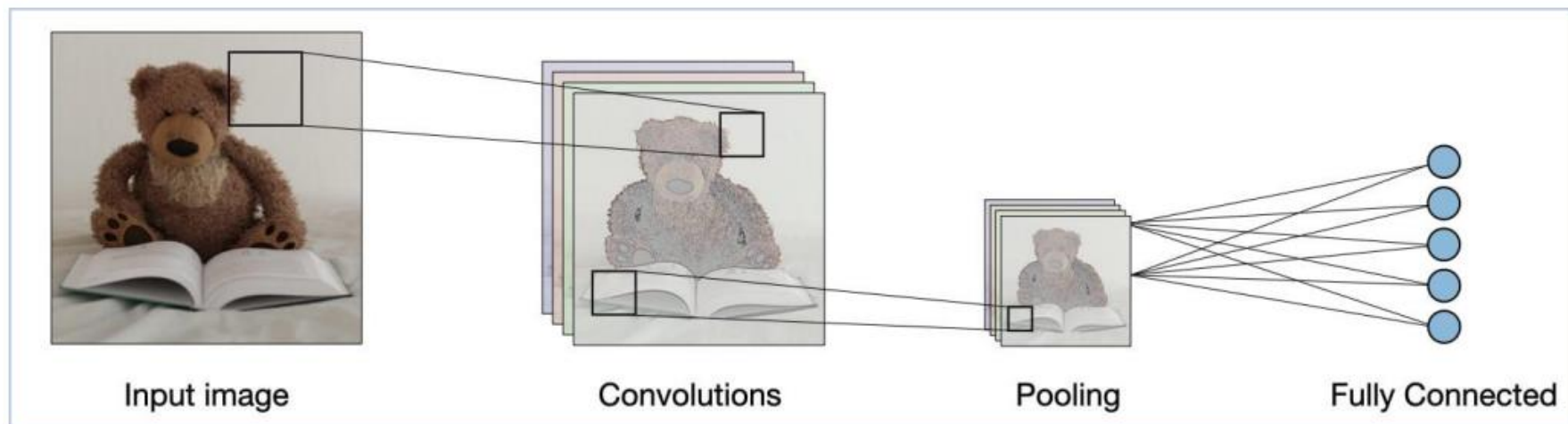
1

CNN

# 1 CNN(Convolutional Neural Network)

- CNN

## CNN이란?



데이터의 **공간 정보** 보존

# 1 CNN(Convolutional Neural Network)

- CNN

CNN이란?

CNN 모델의 3가지 특별한 층(Layer)

Convolution  
Layer

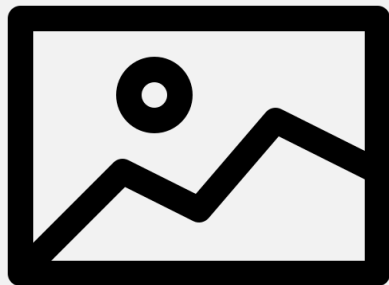
Pooling  
Layer

Fully  
Connected  
Layer

# 1 CNN(Convolutional Neural Network)

- CNN

CNN은 공간 정보가 데이터의 값만큼  
중요한 정보가 되는 형태의 데이터에 대해 아주 강력



이미지 데이터



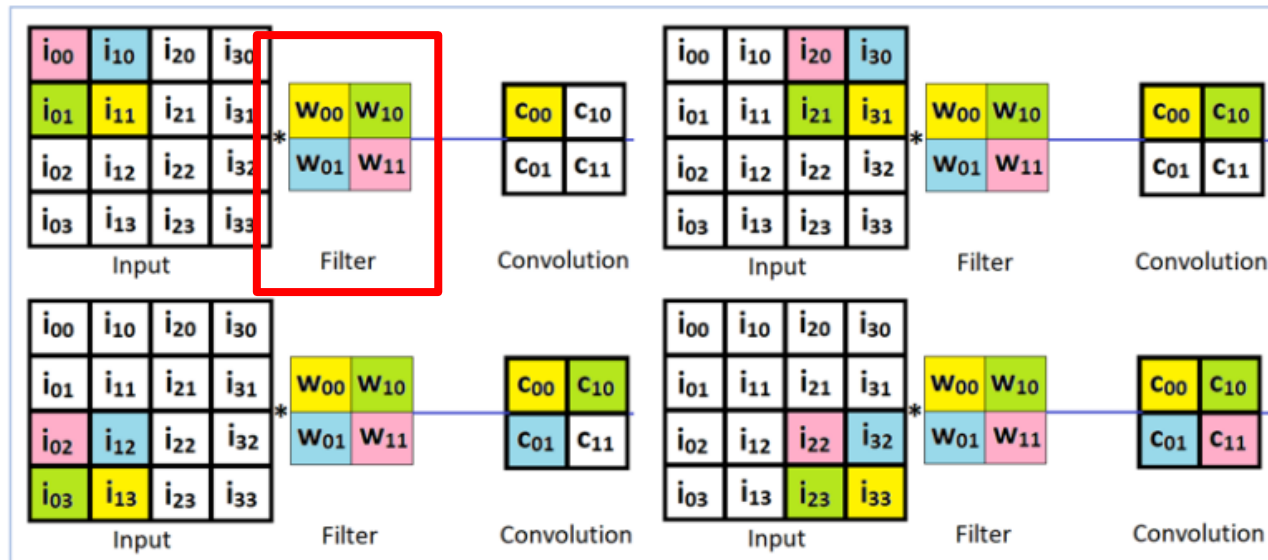
영상 데이터

# 1 CNN(Convolutional Neural Network)

## ● Convolution Layer

### Filter

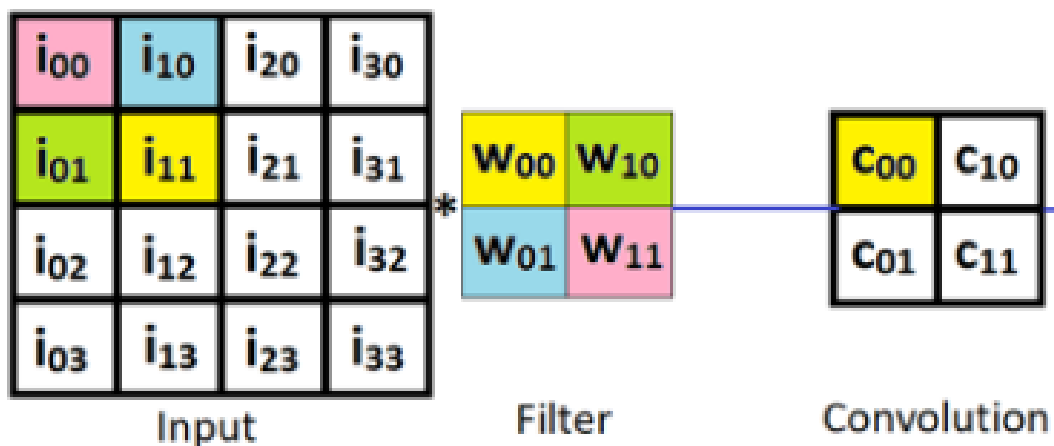
- Input에서 자신의 사이즈만큼의 부분과 가중합하는 역할
  - 순회하면서 Input과 가중합



# 1 CNN(Convolutional Neural Network)

- Convolution Layer

## Convolution Layer의 연산



$$i_{00} \times w_{11} + i_{10} \times w_{01} + i_{01} \times w_{10} + i_{11} \times w_{00} = c_{00}$$

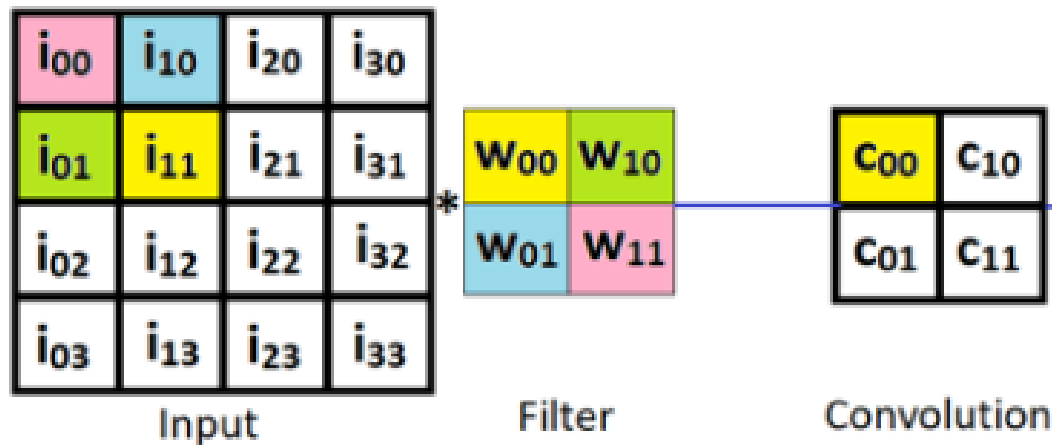
이론적으로 Convolution 연산은 위 수식과 같은 형태



# 1 CNN(Convolutional Neural Network)

- Convolution Layer

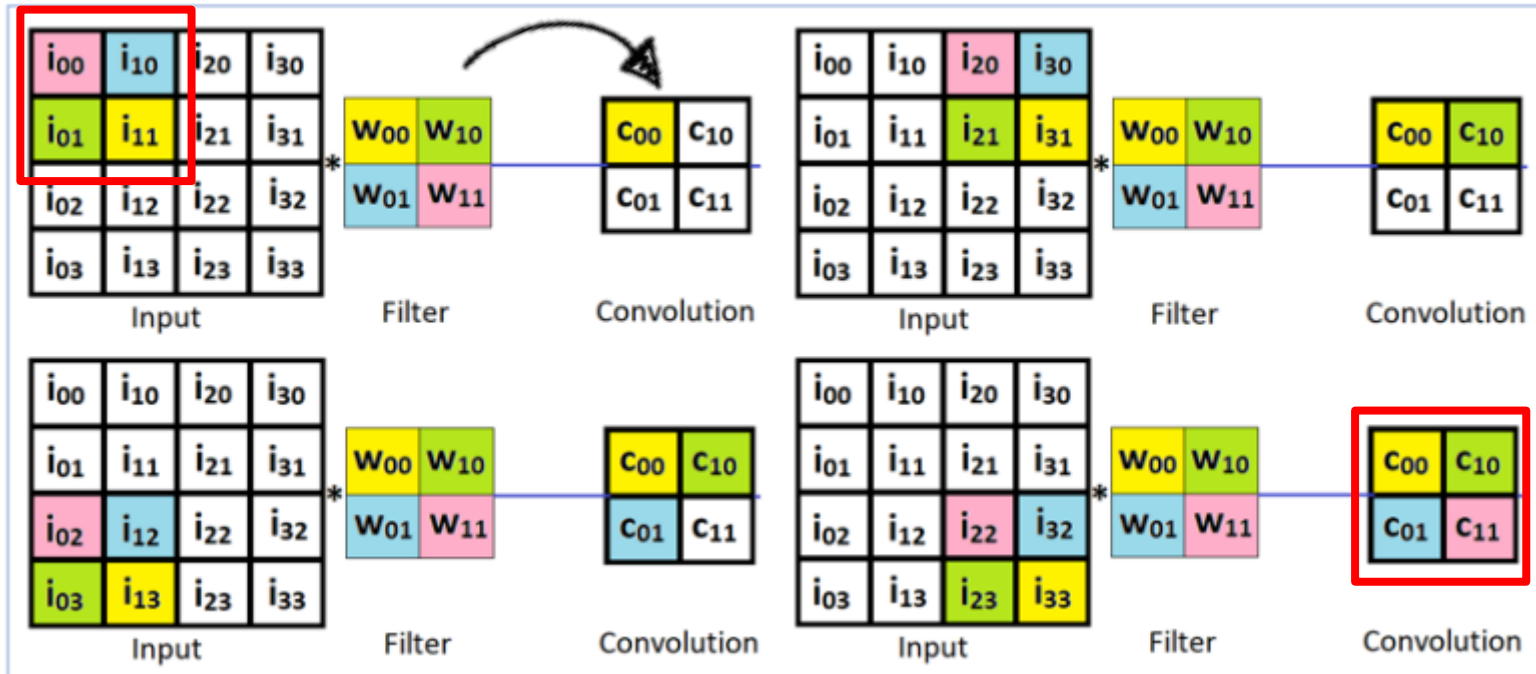
## Cross - Correlation



- 위 그림을 180도 뒤집은 형태
- Pytorch나 tensorflow 등에서 이루어짐
- 두 행렬을 곱할 때 같은 위치에 있는 원소끼리 곱하는 것

# 1 CNN(Convolutional Neural Network)

- Convolution Layer



Feature Map : 반환된 결과

# 1 CNN(Convolutional Neural Network)

- Feature Map

## Feature Map

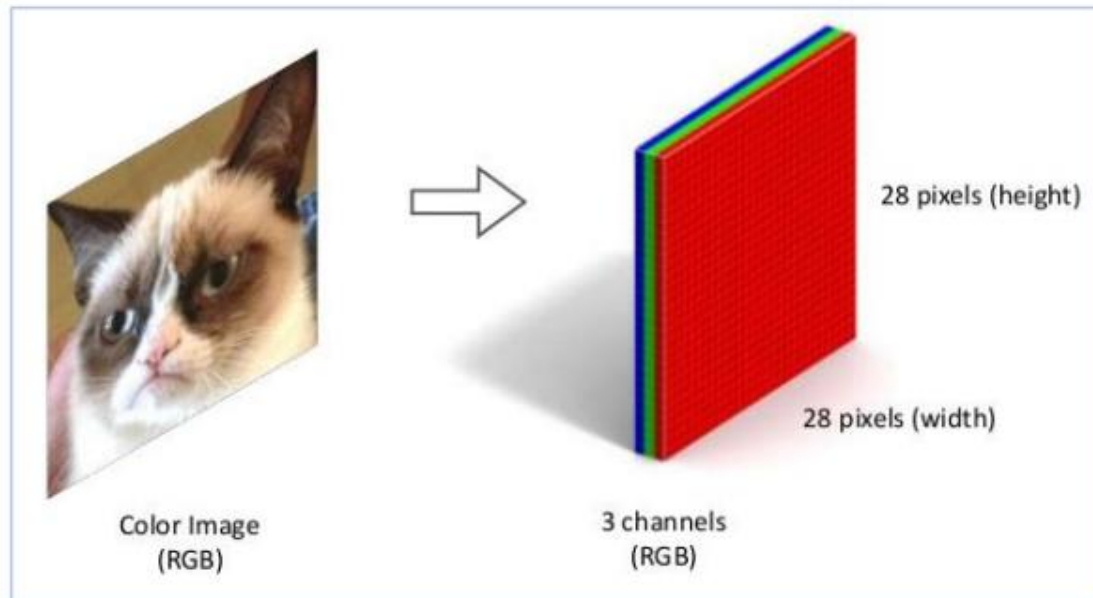
하이퍼 파라미터	Pytorch 표현	역할
Input Channel	in_channels	입력 데이터의 채널 개수 지정
Output Channel	out_channels	출력의 채널 개수 지정
필터 사이즈	kernel_size	필터의 크기 지정
Stride	stride	필터의 이동 간격 지정
Padding	padding	입력 데이터 주변에 붙일 padding 의 수 지정

# 1 CNN(Convolutional Neural Network)

- Input Channel

## Input Channel

- 몇 개의 channel 이 input 으로 한 번에 들어오는가
  - 대부분의 이미지 데이터의 경우 input channel 이 3



R 채널, G 채널, B 채널 -> (3,28,28)

# 1 CNN(Convolutional Neural Network)

- Output Channel

## Output Channel

- 하나의 입력 데이터에 대해 Conv. Layer가 반환하는 **channel의 수**
  - Feature Map의 channel의 수 = Out channel

Feature Map의  
채널이 10개  
각 채널의 사이즈가 (32,32)



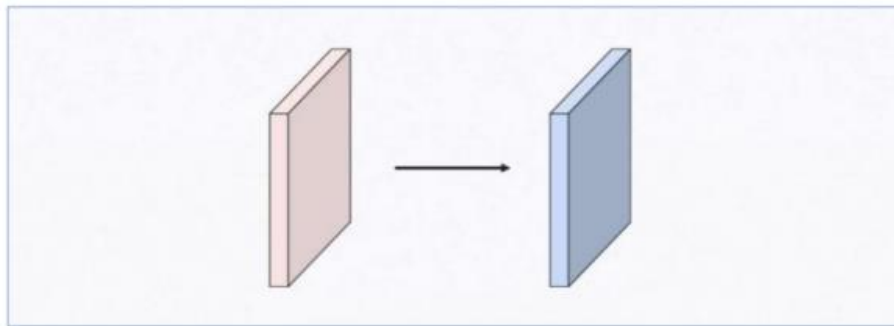
(**10**,32,32)

# 1 CNN(Convolutional Neural Network)

- 필터 사이즈

## 필터 사이즈란?

: 필터의 크기를 얼마나 크게 설정한 것인가



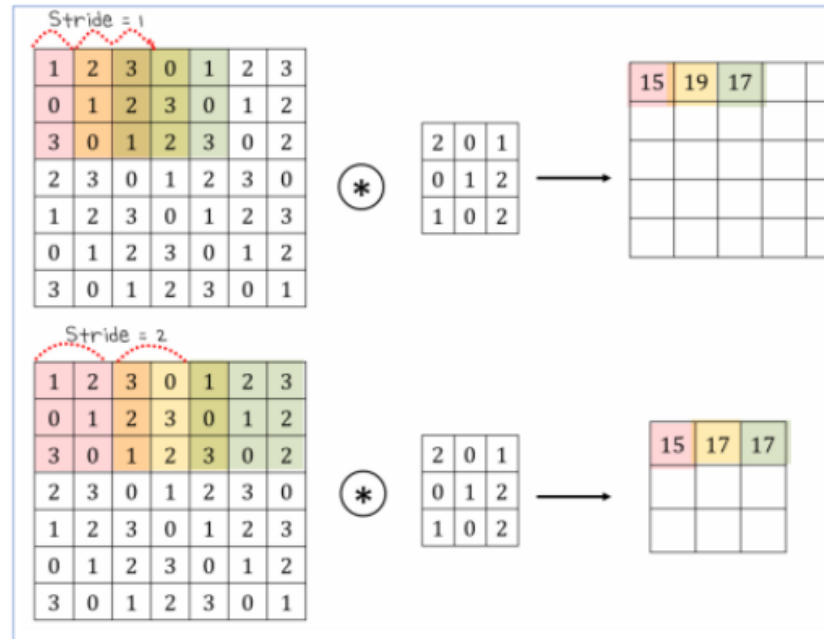
필터의 크기가 클수록 반환되는 Feature Map의 크기는 작아짐

# 1 CNN(Convolutional Neural Network)

- Stride

## Stride란?

: 필터가 이동하는 간격



Stride가 작으면 Feature Map의 크기 커짐

# 1 CNN(Convolutional Neural Network)

- Padding

## Padding

0	0	0	0	0	0				
0									
0									
0									
0									

회색 칸들에 새로운 값들을 넣어주어  
가장자리의 값들도 **필터를 여러 번**  
**통과**할 수 있는 방법



# 1 CNN(Convolutional Neural Network)

- Feature Map의 크기

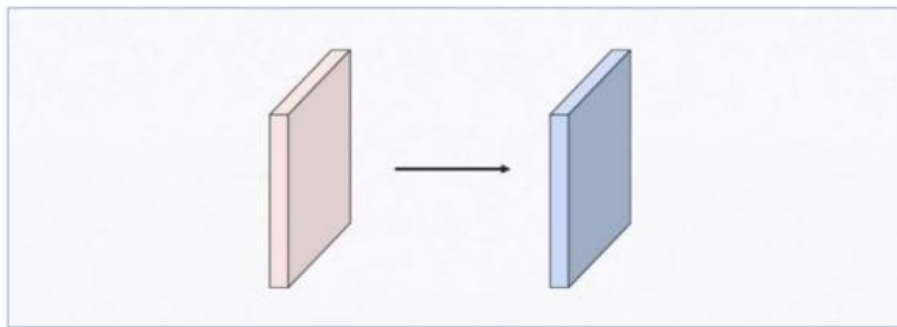
$$O_n = \frac{I_n + 2P - F}{S} + 1$$

- $O_n$  : 출력의 가로길이
- $I_n$  : 입력의 가로길이
- $P$  : Padding의 크기
- $F$  : 필터의 가로 길이
- $S$  : Stride

# 1 CNN(Convolutional Neural Network)

- Feature Map의 크기

## Feature Map 예시



- $I_n$  : 입력데이터의 사이즈 (3,32,32)
- $F$  : Conv. Layer에 (4,4)필터 10개
- $S$  : Stride=2
- $P$  : Padding = 1

$$O_n = \frac{I_n + 2P - F}{S} + 1 = \frac{32 + 2 \times 1 - 4}{2} + 1 = 16$$

# 1 CNN(Convolutional Neural Network)

- Pooling Layer

## Pooling Layer란?

: 입력 이미지의 중요한 특징들을 더욱 강조하는 동시에 데이터의 크기를 줄이는 것에 더 중점을 둔 Layer

Convolution  
Layer

가중치 존재

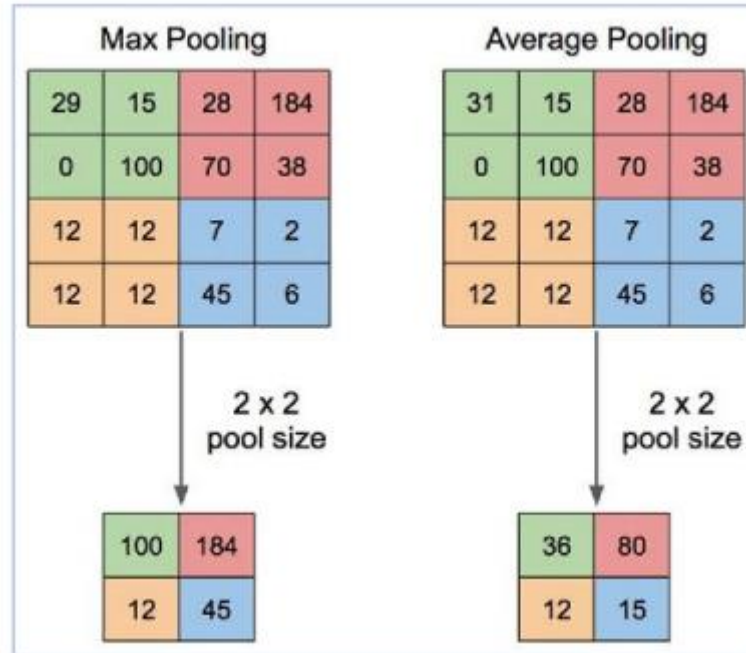
Pooling  
Layer

가중치 없음

# 1 CNN(Convolutional Neural Network)

- Pooling Layer

## Pooling Layer란?

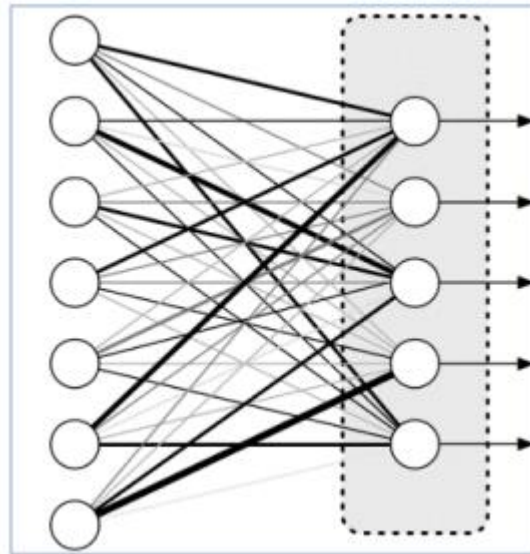


- 가중치 없음
- 입력의 특징을 가져오면서 크기를 줄임
- Max Pooling, Average Pooling, Min Pooling

# 1 CNN(Convolutional Neural Network)

- Fully Connected Layer

## Fully Connected Layer



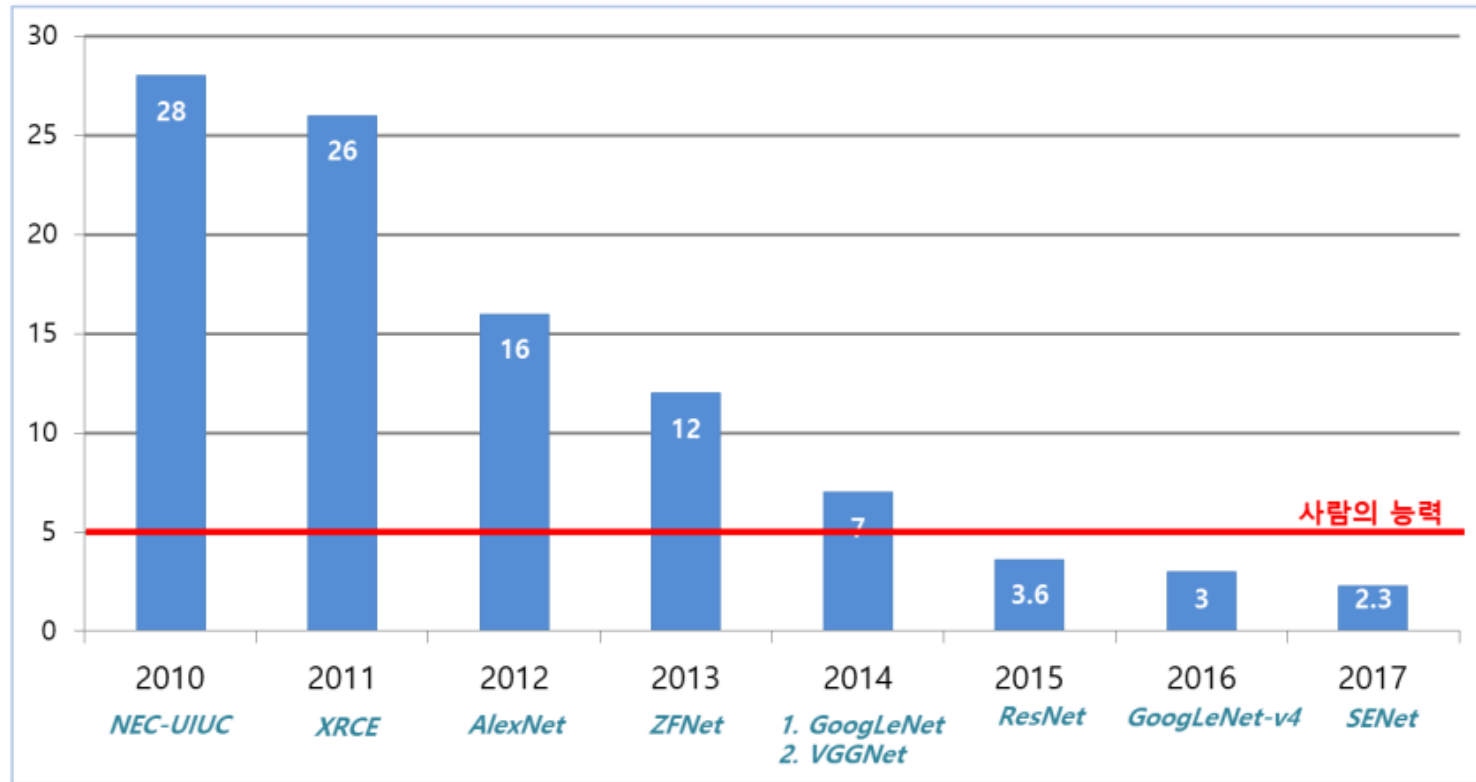
최종 단계에서 라벨 예측을 진행할 때 사용되는 Layer  
벡터로의 변환이 필수적

# 2

## CNN의 발전 과정

## 2 CNN의 발전 과정

- CNN의 발전 과정



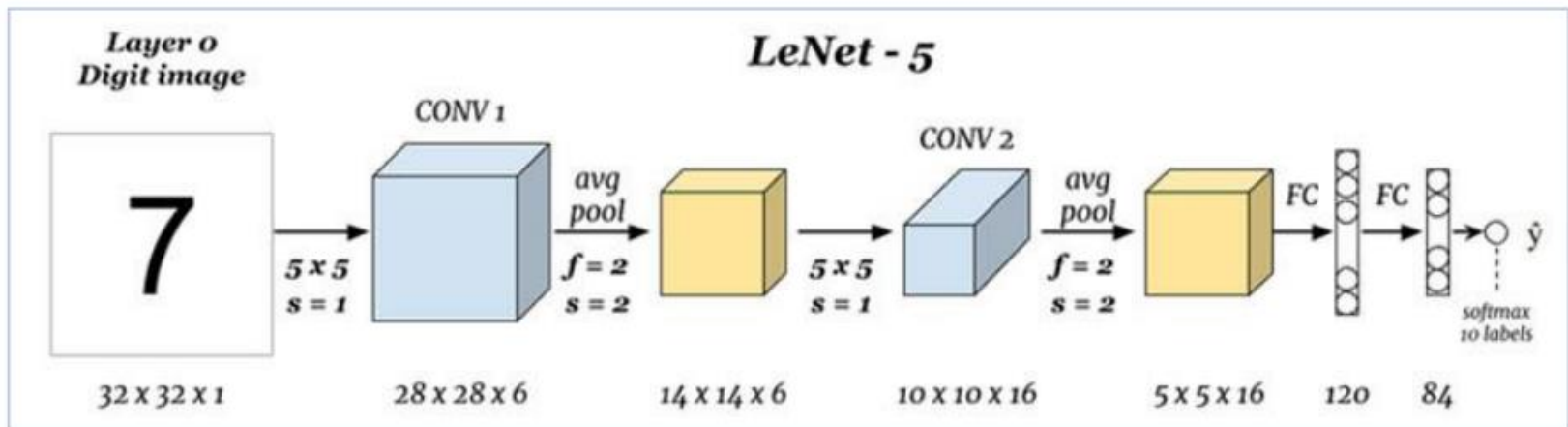
Lenet-5, AlexNet, GoogLeNet, VGGNet, ResNet

## 2 CNN의 발전 과정

### ● LeNet-5

#### LeNet-5란?

- 맨 처음 제안된 CNN 모델
- (1,32,32)의 입력을 Layer들에 통과시켜 0~9까지의 손으로 쓴 숫자를 분류하는 모델





## 2 CNN의 발전 과정

### ● LeNet-5

#### LeNet-5의 특징

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

- 모든 layer의 활성화 함수로 tanh를 사용
- Tanh 함수는 미분값이 0에 수렴

## 2 CNN의 발전 과정

- LeNet-5

### LeNet-5의 특징

Layer	# filters / neurons	Filter size	Stride	Size of feature map	Activation function
Input	-	-	-	32 X 32 X 1	
Conv 1	6	5 * 5	1	28 X 28 X 6	tanh
Avg. pooling 1		2 * 2	2	14 X 14 X 6	
Conv 2	16	5 * 5	1	10 X 10 X 16	tanh
Avg. pooling 2		2 * 2	2	5 X 5 X 16	
Conv 3	120	5 * 5	1	120	tanh
Fully Connected 1	-	-	-	84	tanh
Fully Connected 2	-	-	-	10	Softmax

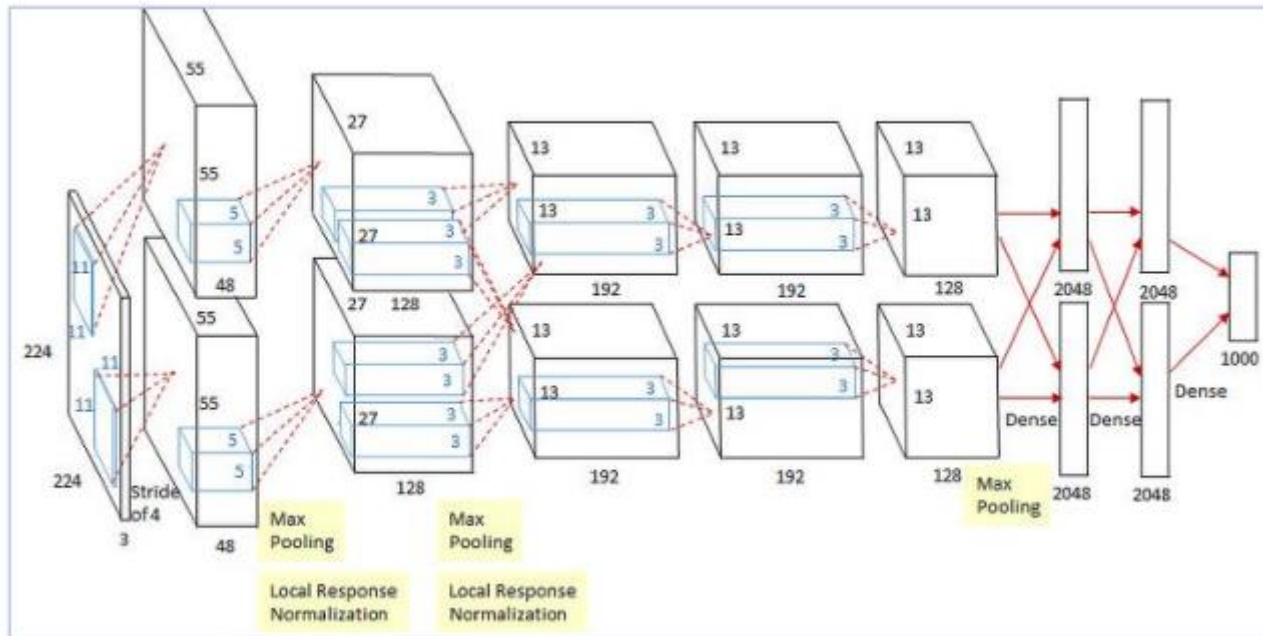
현재의 CNN에서는 잘 사용하지 않음

## 2 CNN의 발전 과정

- AlexNet

### AlexNet의 특징

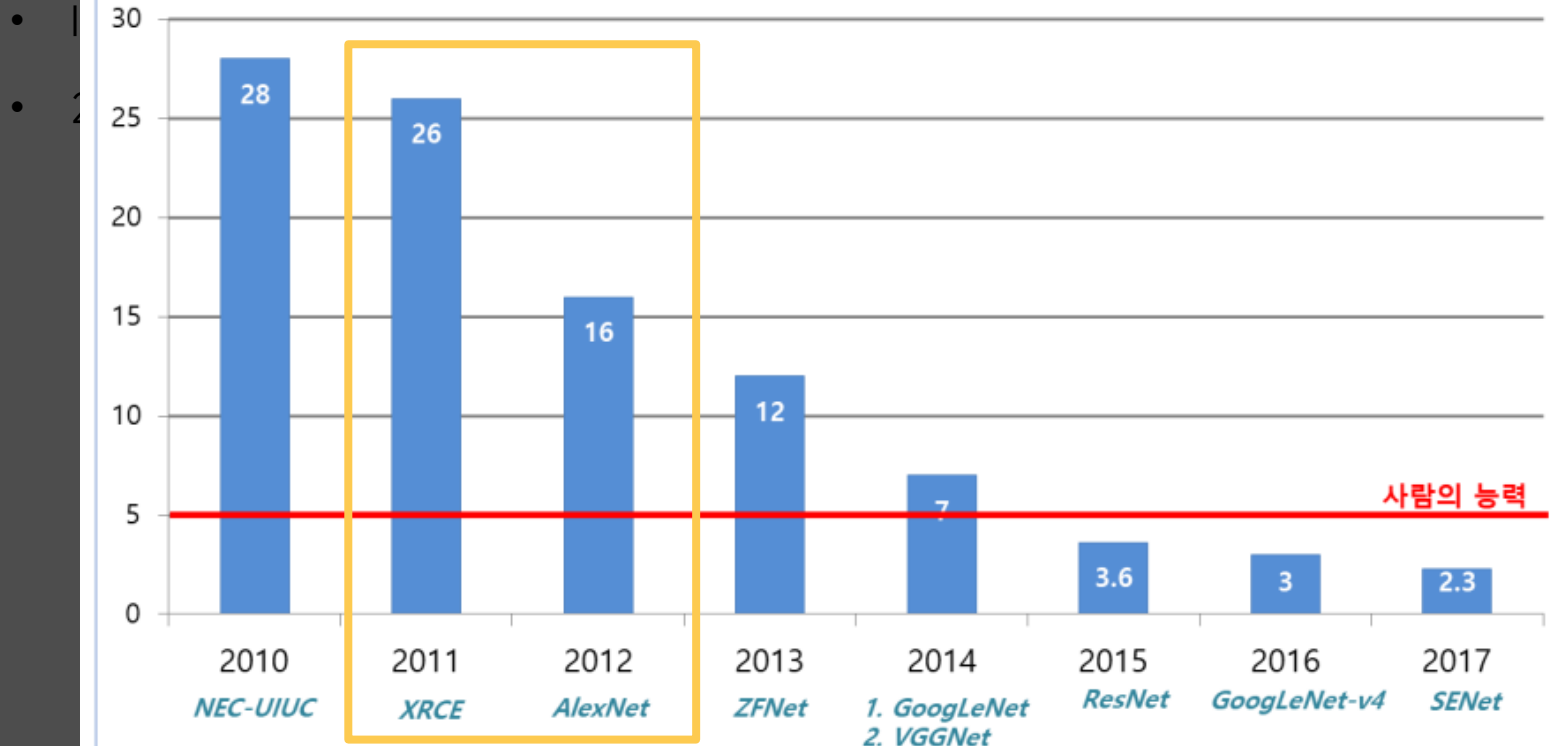
- ILSVRC-2012에서 압도적 1등 차지 - 오류율이 큰 폭으로 감소
- 224 \* 224의 컬러 이미지 처리를 위해 병렬 처리



## 2 CNN의 발전 과정

- AlexNet

### AlexNet의 특징



Local Response  
Normalization

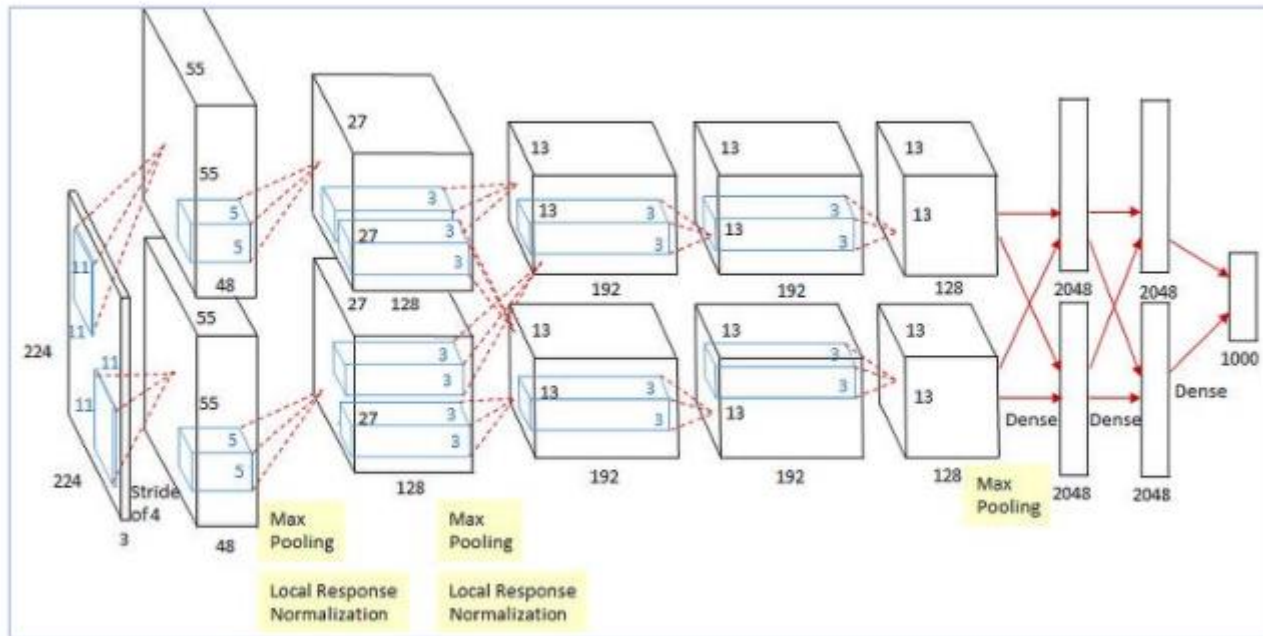
Local Response  
Normalization

## 2 CNN의 발전 과정

### ● AlexNet

#### AlexNet의 특징

- ILSVRC-2012에서 압도적 1등 차지 - 오류율이 큰 폭으로 감소
- 224 \* 224의 컬러 이미지 처리를 위해 병렬 처리 ➡ 2개의 GPU를 병렬적으로 이용

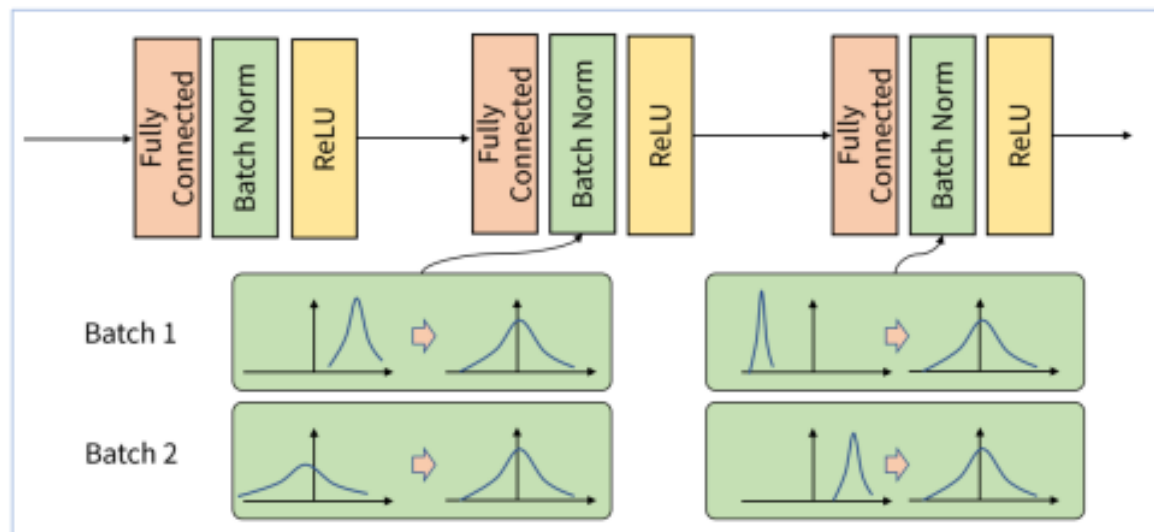


## 2 CNN의 발전 과정

- 배치 정규화

### 배치 정규화

- 데이터 분석의 전처리 단계에서 시행하는 데이터 **Scaling** 과정의 **Normalization**과 유사  
속성별로 scale을 평균과 분산으로 정규화 해주는 과정

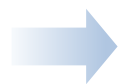


## 2 CNN의 발전 과정

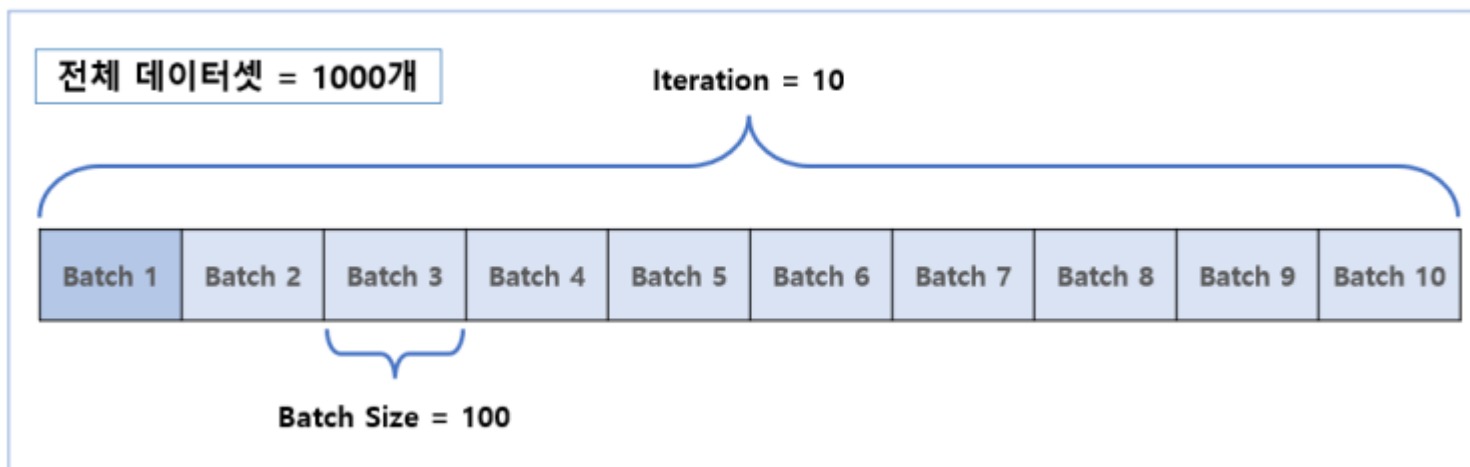
### ● 배치 정규화

#### 배치 정규화

- 배치의 크기에 따라 Iteration의 크기가 결정
- 모집단의 분포와 표본의 분포가 같지 않은 것처럼, 배치마다 데이터의 분포가 달라질 수 있음



Batch 별 데이터를 평균과 분산으로 정규화 해주는 것

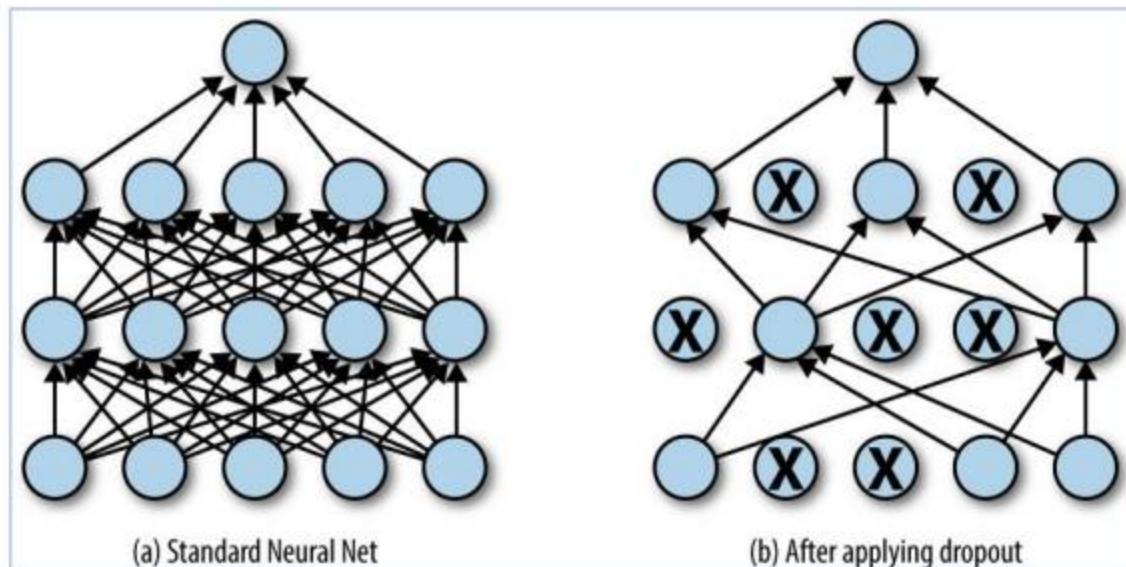


## 2 CNN의 발전 과정

### ● Dropout

#### Dropout

- Batch마다 학습할 때 일정한 비율의 노드를 **버리고** 학습하는 방법
  - 여러 모델을 **양상불** 한 것과 **유사한 효과**
  - 과적합을 **방지**할 수 있는 방법

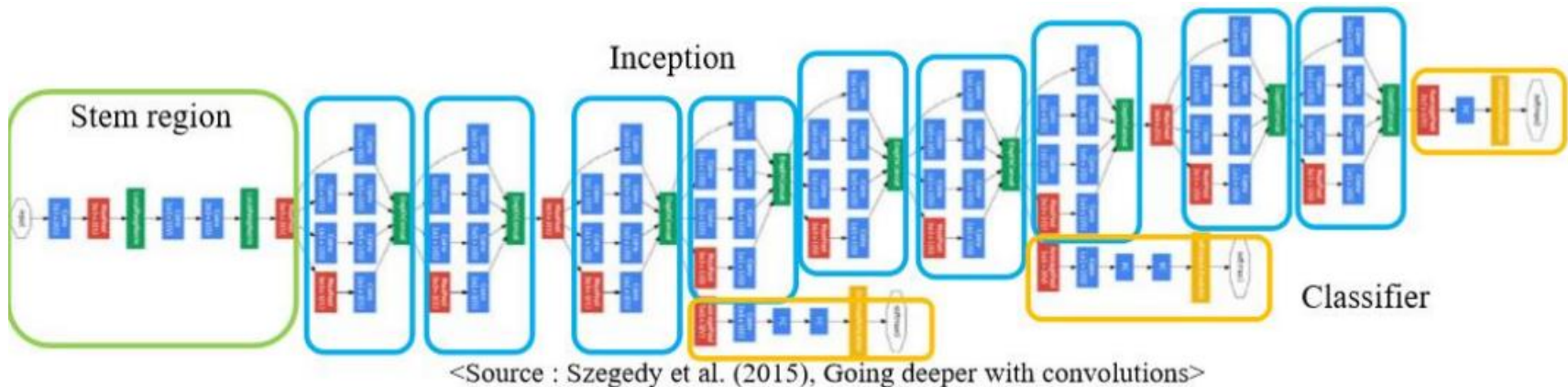




## 2 CNN의 발전 과정

- GoogLeNet

### GoogLeNet의 구조와 특징



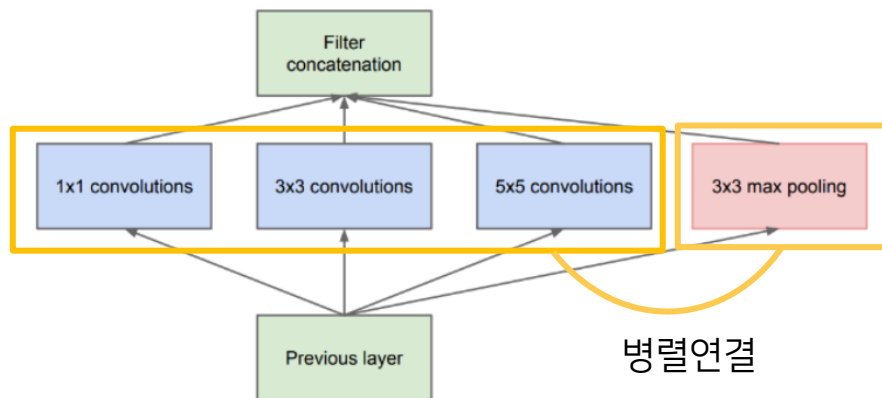
- 모델의 **중간마다** 역전파 실행
- Inception 모듈 → AlexNet 파라미터의 1/12로 축소
- FC Layer 사용 X

## 2 CNN의 발전 과정

- GoogLeNet

### Inception 모듈

: 이미지 데이터의 특성상 가까운 픽셀끼리는 상관성이 높고 먼 픽셀끼리는 상관성이 낮다는 점을 반영



(a) Inception module, naïve version

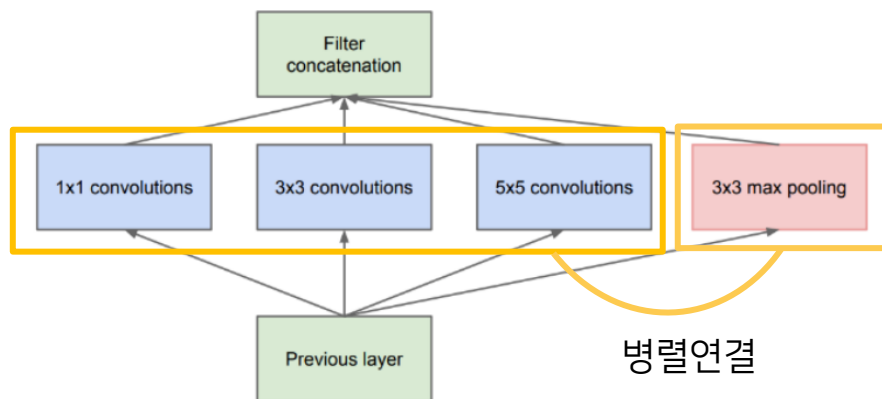
문제점: 연산 ↑

## 2 CNN의 발전 과정

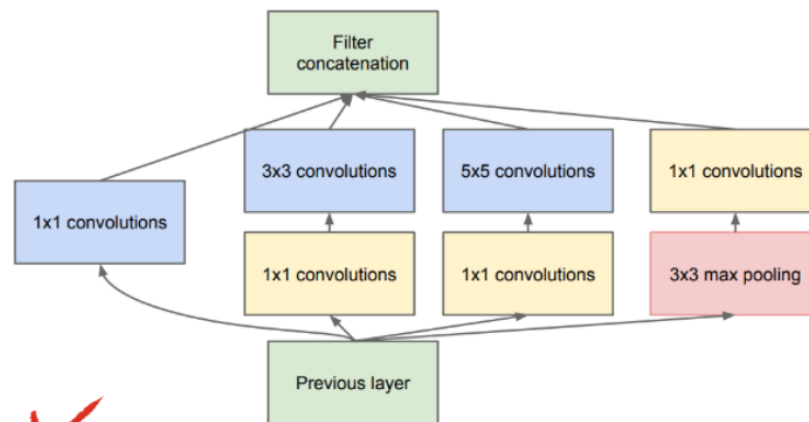
### ● GoogLeNet

#### Inception 모듈

: 이미지 데이터의 특성상 가까운 픽셀끼리는 상관성이 높고 먼 픽셀끼리는 상관성이 낮다는 점을 반영



(a) Inception module, naïve version



(b) Inception module with dimension reductions

문제점 : 연산 ↑

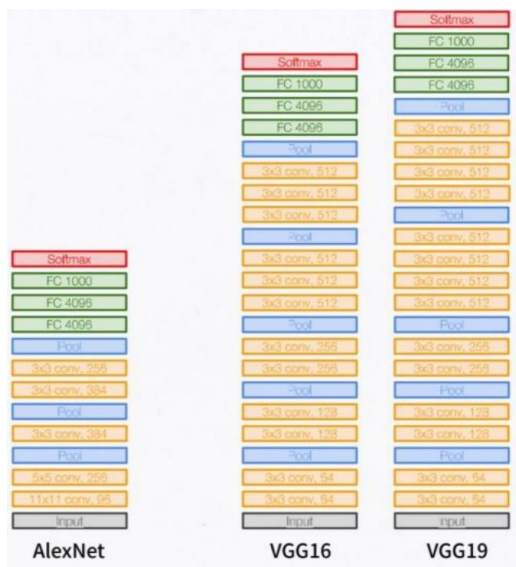


- (1,1) Conv.Layer 각 단계에 추가
- Feature Map의 depth ↓  
(연산 ↓)

## 2 CNN의 발전 과정

### ● VGGNet

#### VGGNet의 구조와 특징



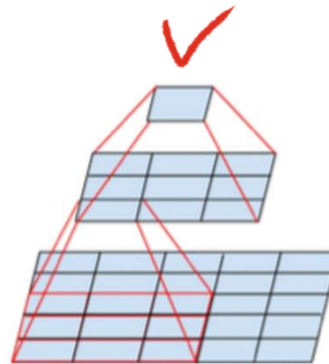
더 많은 layer 사용

How?

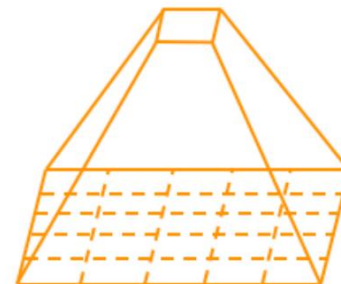
파라미터 수:  
 $2 \times 3^2 \times \text{채널 수}$

<

$5^2 \times \text{채널 수}$



two successive  
3x3 convolutions



5x5 convolution

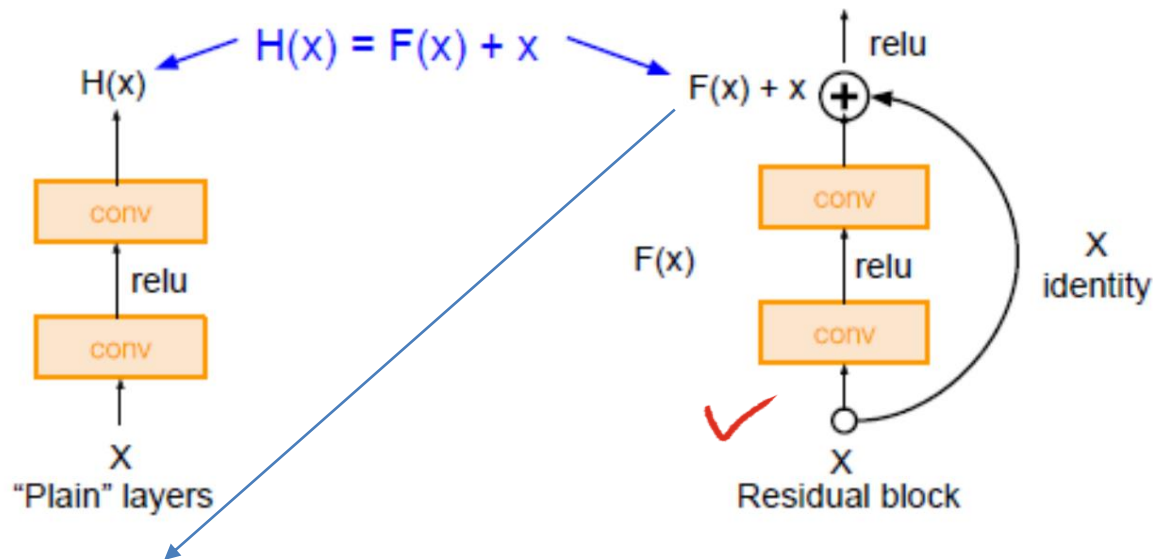
- 작은 필터 여러 번 사용
- 파라미터 수 감소 ↓ 비선형성 ↑

## 2 CNN의 발전 과정

- ResNet

### ResNet의 구조와 특징

: Residual Learning을 통해 더 깊은 층을 쌓은 모델



$$H(x) =$$

$F(x)$ : 이전 layer 결과를 입력으로 사용

+

$x$ : Conv. Layer 이후로 넘겨 그대로 사용

$$\frac{\partial H(x)}{\partial x} = (F(x) + x)' = F'(x) + 1$$

이전에 학습되지 못한  $F(x)$  최적화

3

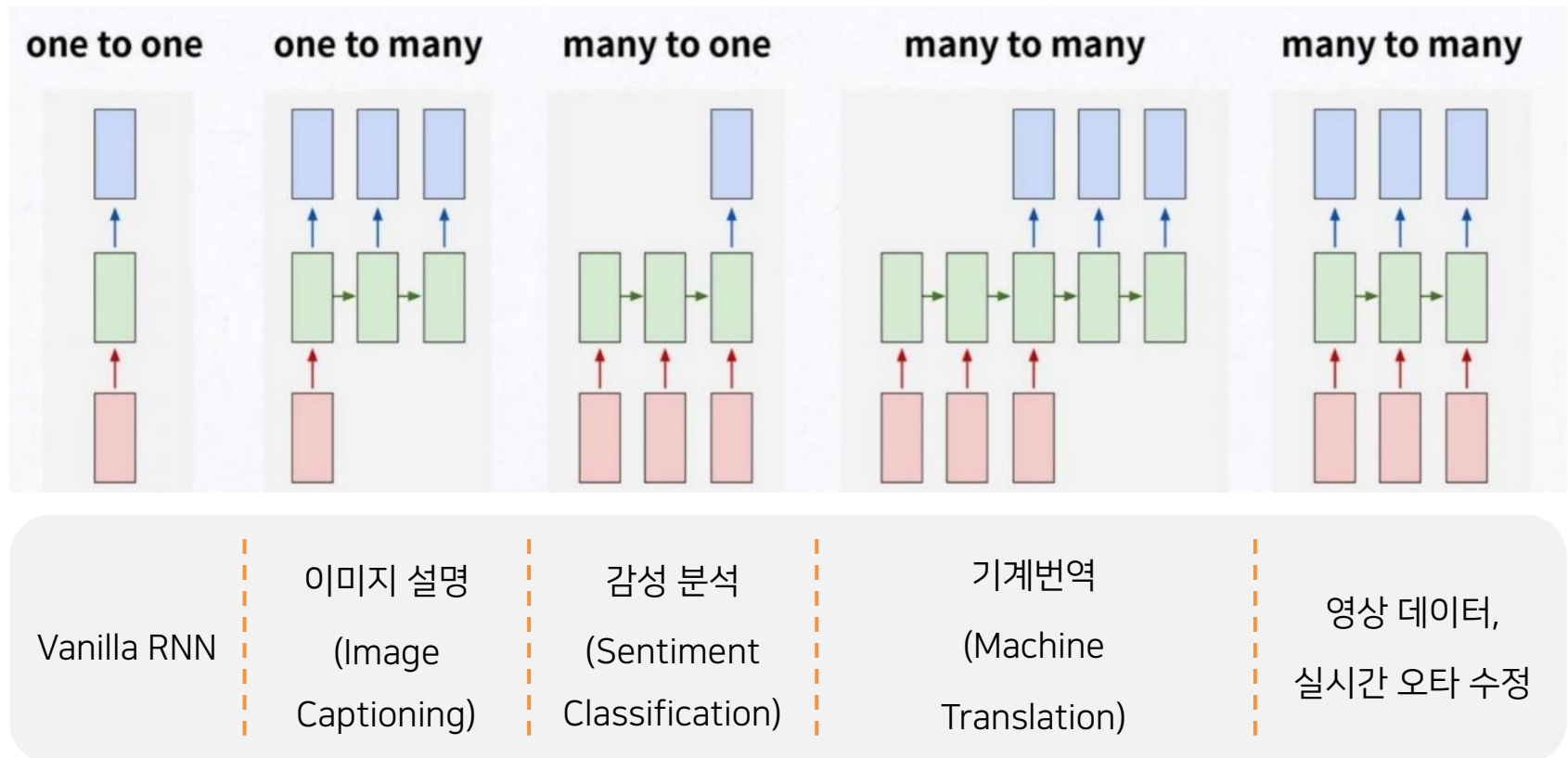
RNN

# 3 RNN (Recurrent Neural Network)

- RNN 모델의 형태

## RNN

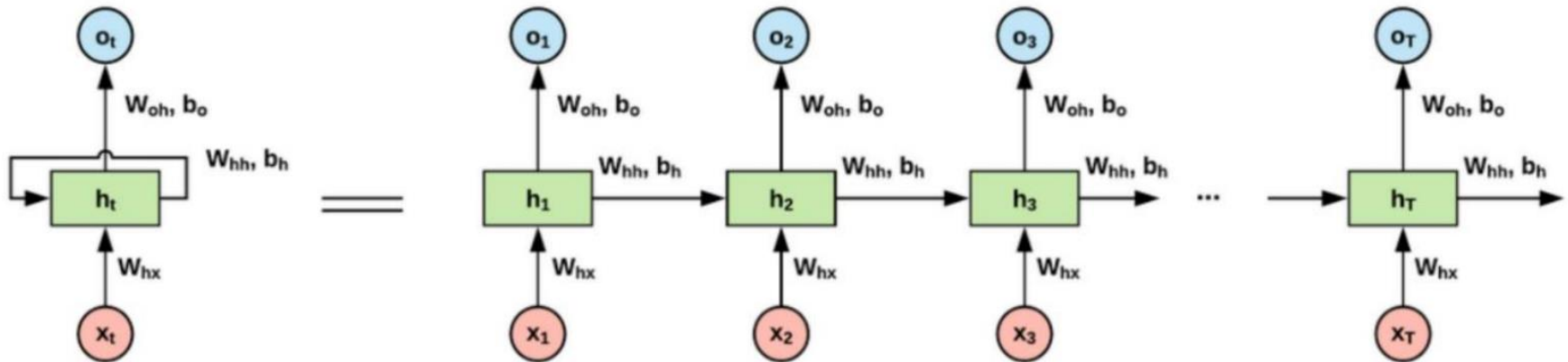
: Sequential Data를 학습할 수 있는 딥러닝 모델



### 3 RNN (Recurrent Neural Network)

- Vanilla RNN

#### Vanilla RNN의 구조와 특징



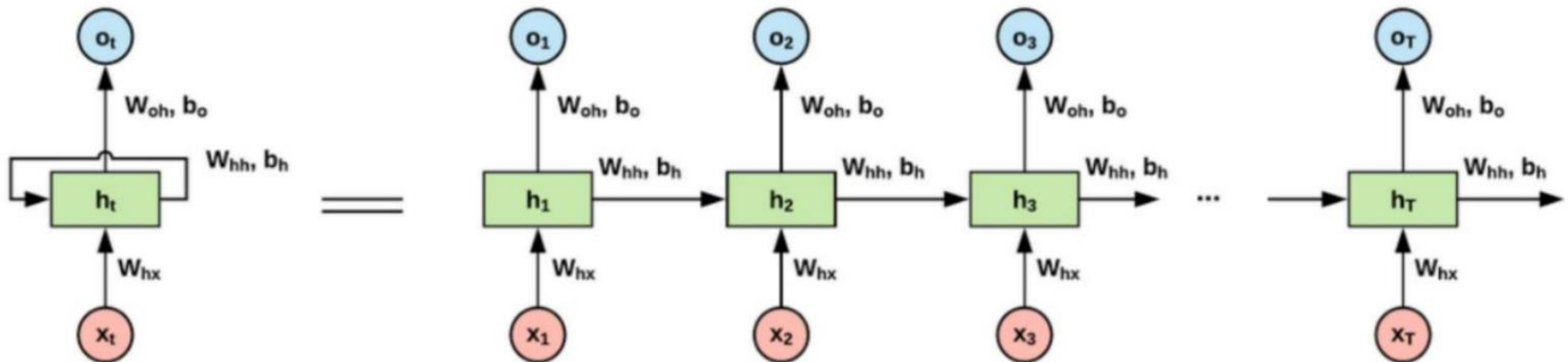
Hidden Layer(Memory Cell) : 처음부터 마지막 시점  $t$ 의 데이터까지 반복해서 사용됨



# 3 RNN (Recurrent Neural Network)

- Vanilla RNN

## Vanilla RNN의 구조와 특징

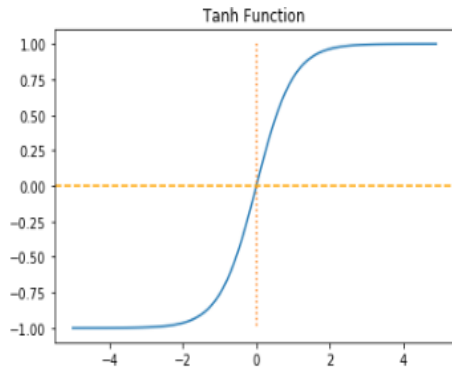


- $x_t$ : 입력
- $h_t$ (Hidden State): Memory Cell 값
  - 각 시점의 입력에 따라 업데이트되며 어떤 데이터를 기억할지 결정

# 3 RNN (Recurrent Neural Network)

## Vanilla RNN

### Hidden State



$$h_t = \tanh(W_x \times x_t + W_h \times h_{t-1})$$

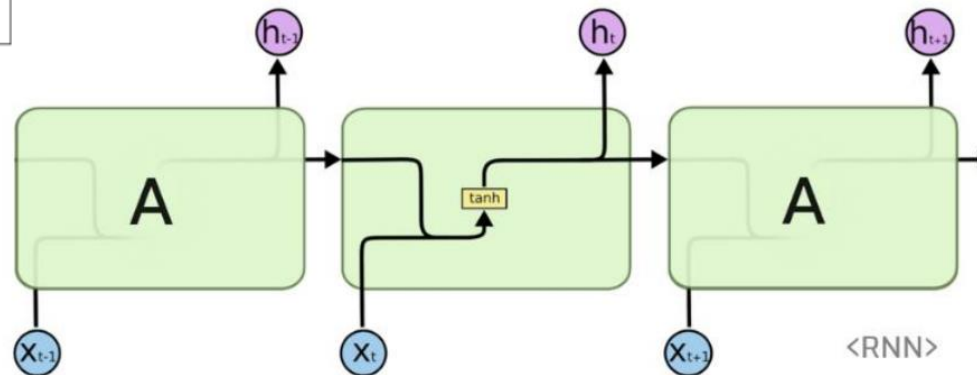
$$o_t = f(W_o h_t)$$

Vanilla RNN의 가중치

$$W_x: x_t \rightarrow h_t$$

$$W_h: h_t \rightarrow h_{t+1}$$

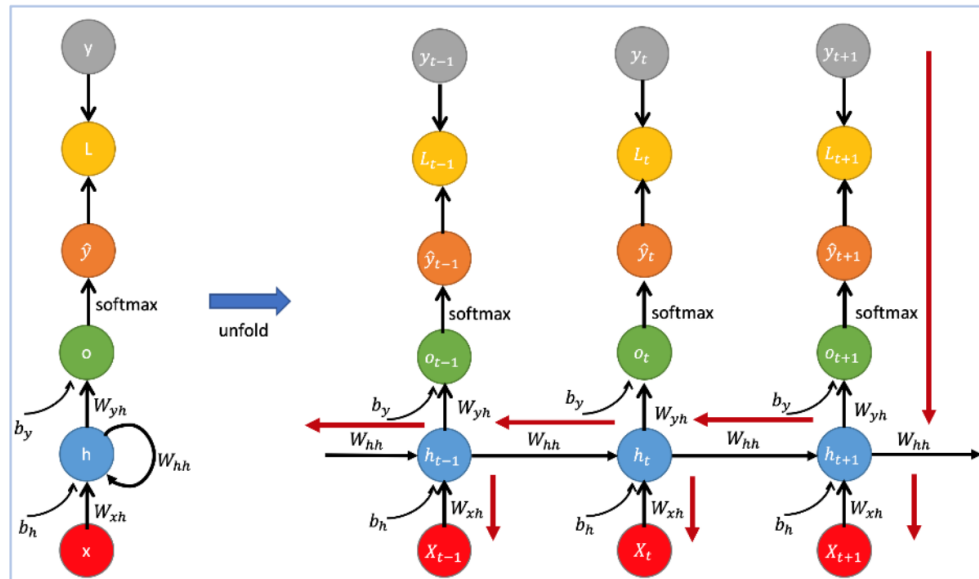
$$W_o: h_t \rightarrow o_t$$



### 3 RNN (Recurrent Neural Network)

- 역전파 (Back Propagation)

#### BPTT (Back Propagation Through Time)

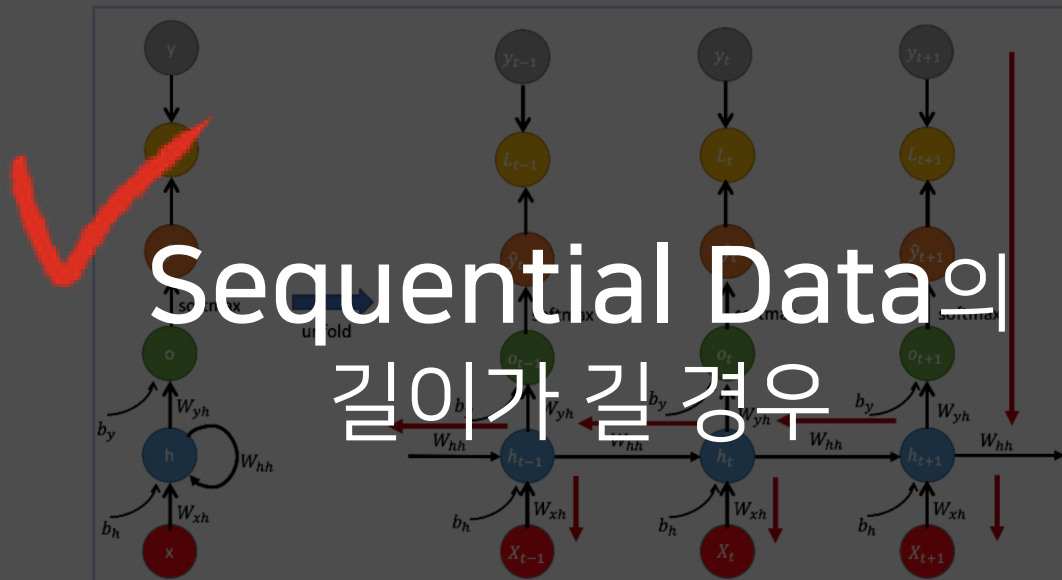


매 시점 gradient 계산 → 첫번째 시점까지 역전파

### 3 RNN (Recurrent Neural Network)

- 역전파 (Back Propagation)

BPTT (Back Propagation Through Time)



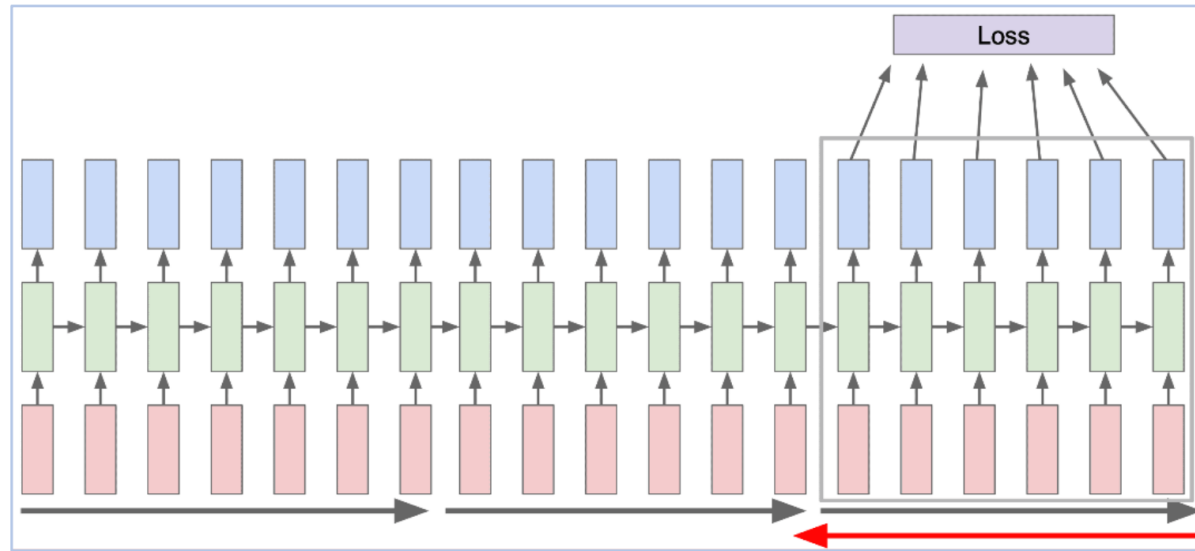
연산량↑ 소요 시간↑

매 시점 gradient 계산 → 첫번째 시점까지 역전파

### 3 RNN (Recurrent Neural Network)

- 역전파 (Back Propagation)

#### Truncated BPTT



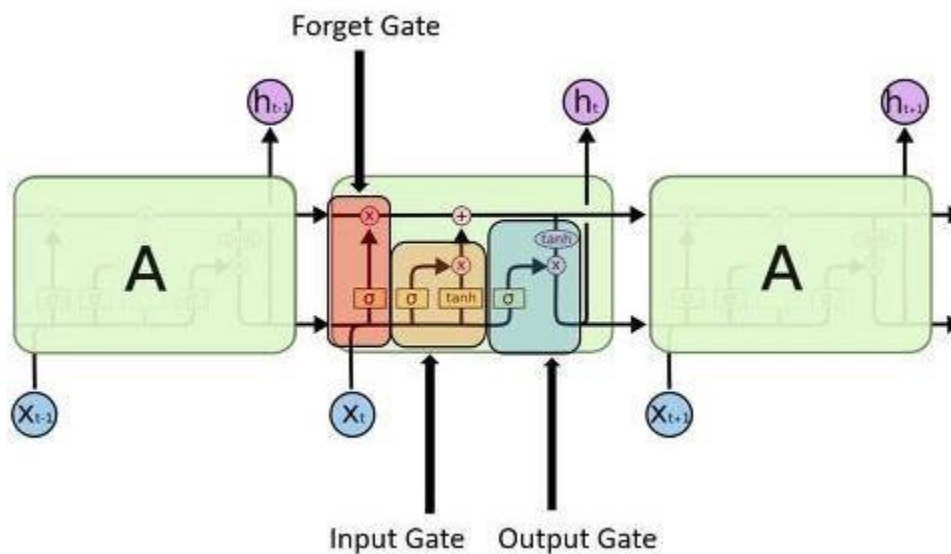
Sequential Data 구간 분리 → BPTT 진행

### 3 RNN (Recurrent Neural Network)

- LSTM(Long Short-Term Memory)

#### LSTM이란?

: 사람의 장기 기억과 단기 기억에 착안하여 만들어진 모델

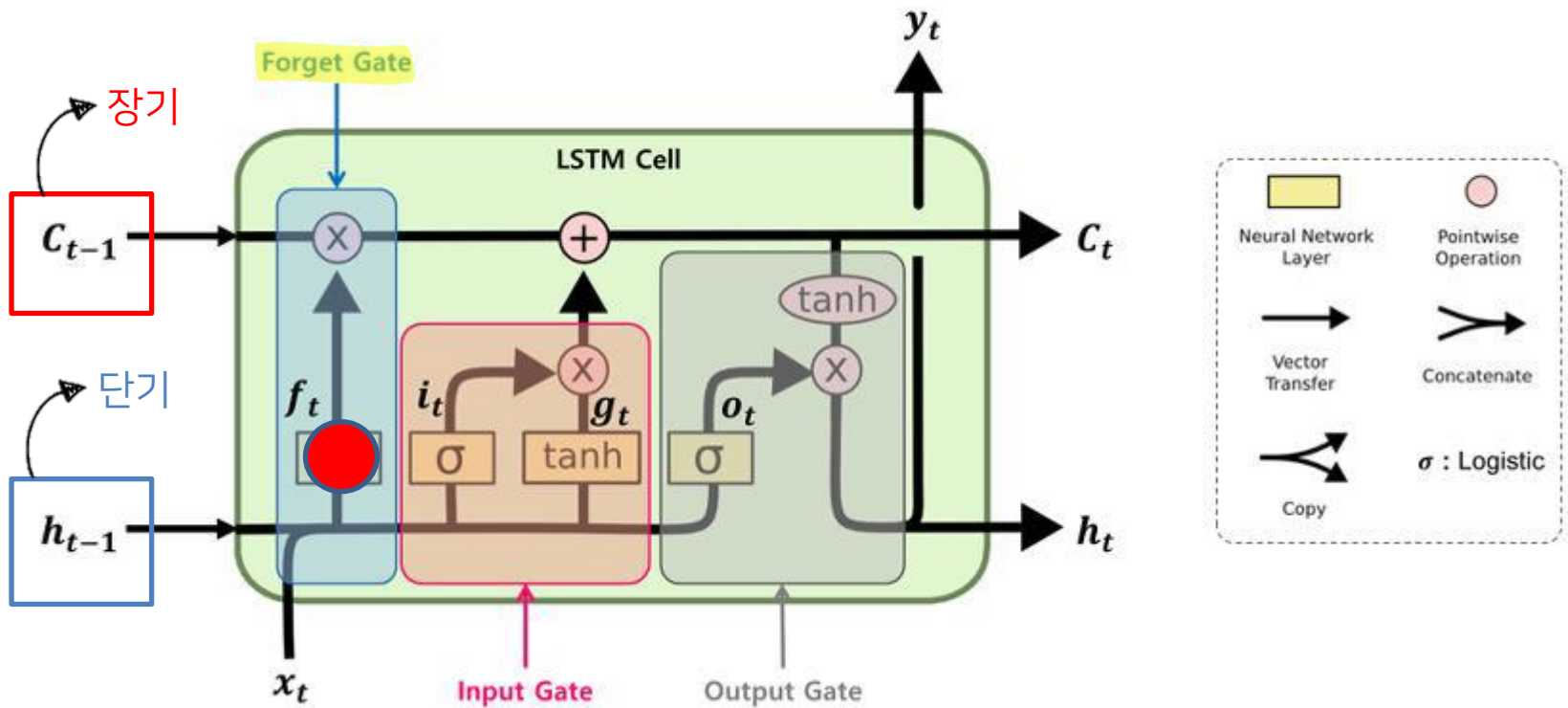


### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Forget Gate

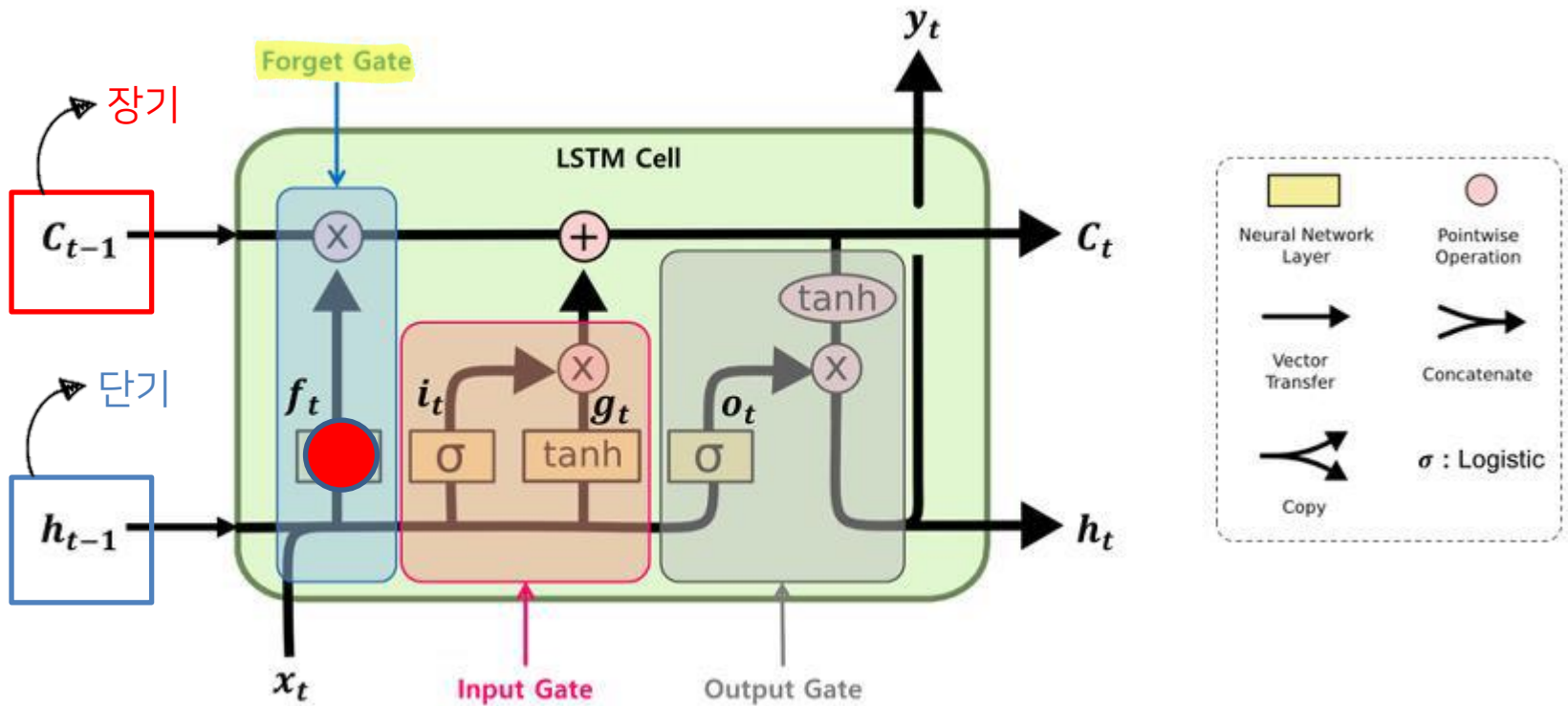
: 과거의 정보를 얼마나 잊을지 결정하는 역할



### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Forget Gate



이전 시점의 Hidden State 와 현재 시점의 입력을 가중치와 곱해 시그모이드 함수에 전달



### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Forget Gate

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

시그모이드 함수

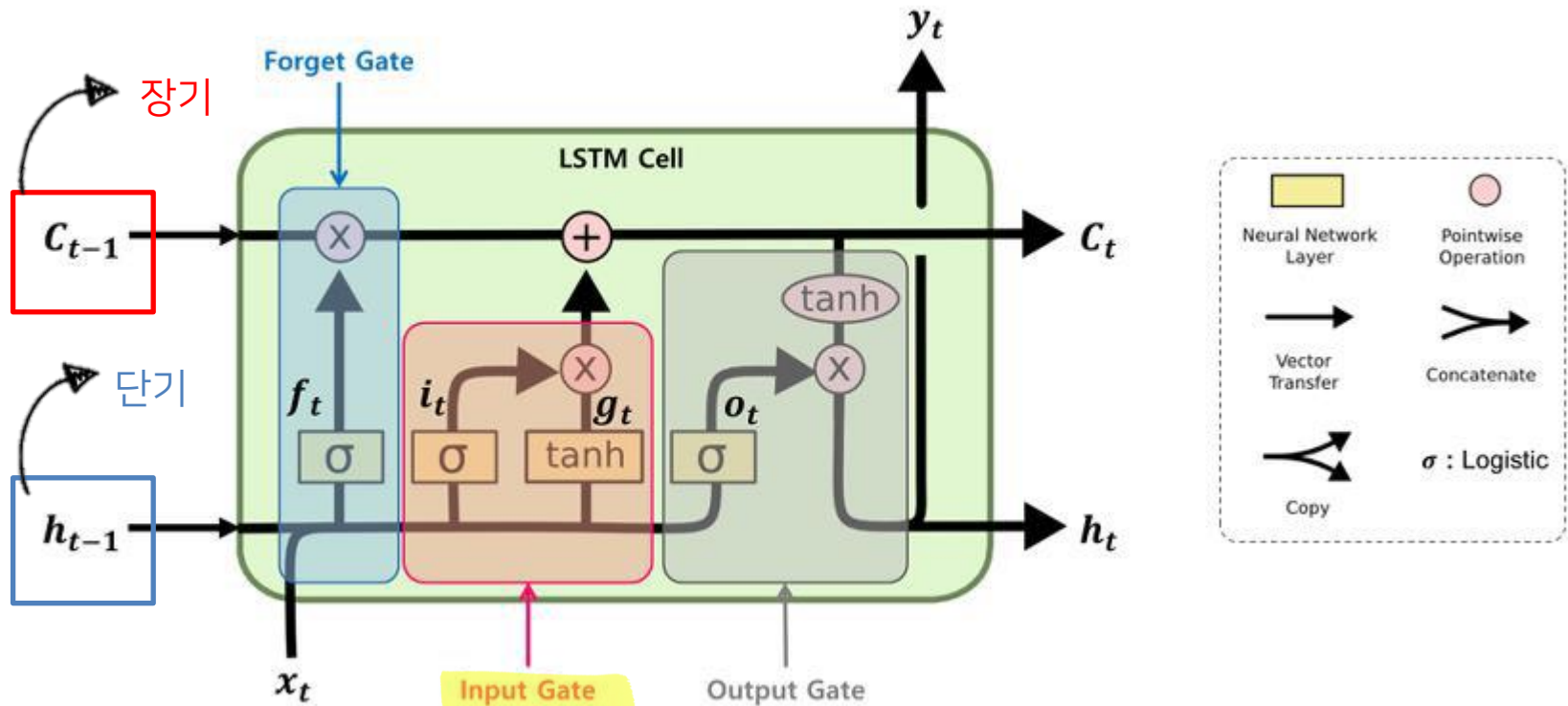
- 활성화 함수는 시그모이드 함수
- 출력이 1에 가까울 수록 이전 시점 정보들을 많이 기억함

### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Input Gate

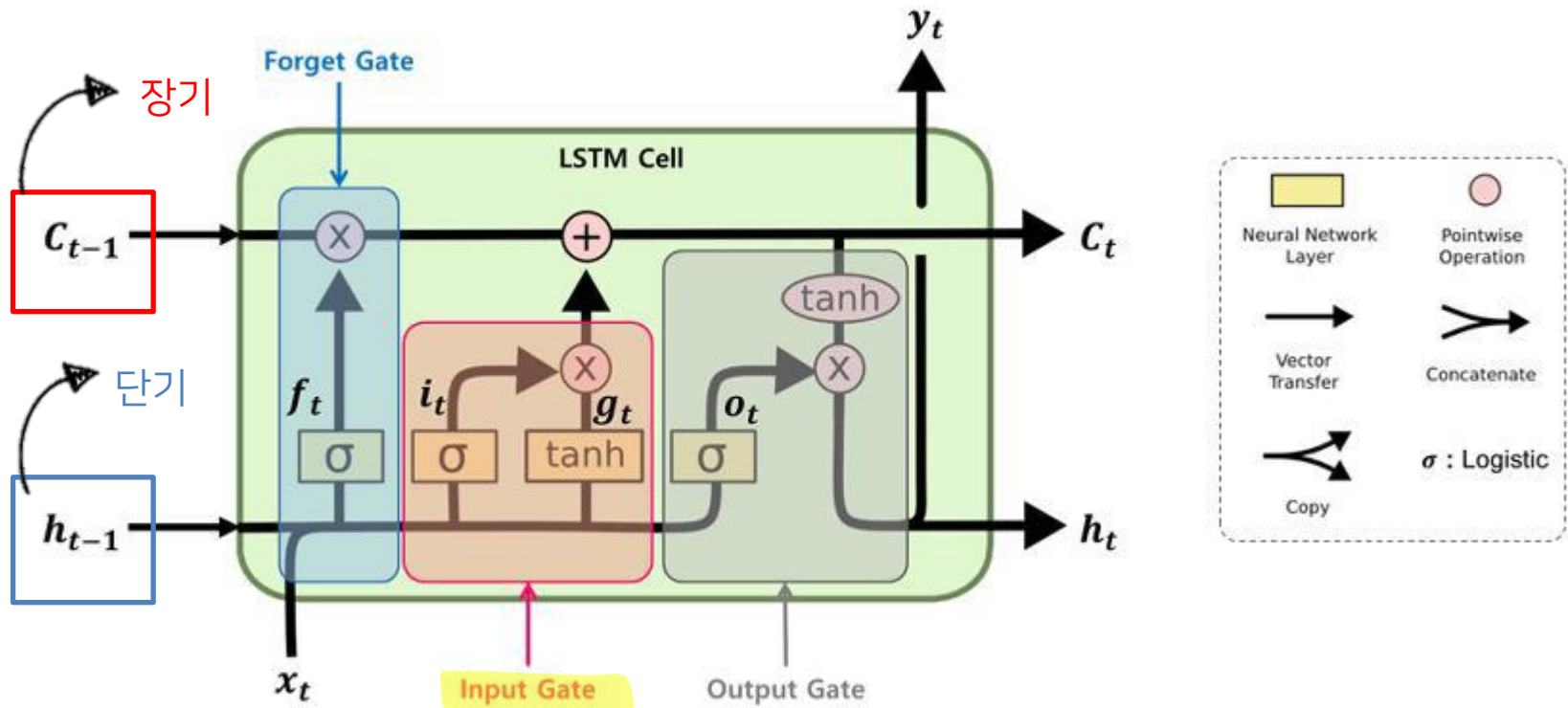
: 현재의 정보를 얼마나 기억할지 결정하는 역할



### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Input Gate



이전 시점의 Hidden State 와 현재 시점의 입력을 가중치와 곱해 시그모이드 또는 tanh 함수에 전달

### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Input Gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$g_t = \tanh(W_g \cdot [h_{t-1}, x_t] + b_g)$$

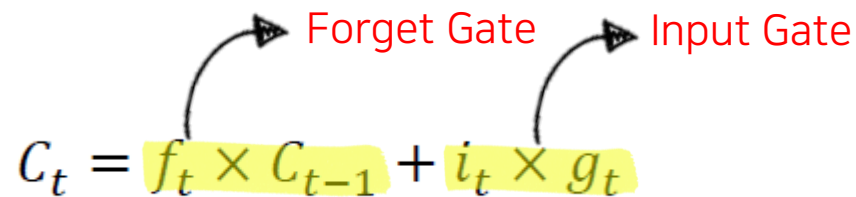
- 활성화 함수는 시그모이드, tanh 함수
- 시그모이드: 이전 시점 정보를 얼마나 Cell State에 전달할 지 결정
- Tanh: 이전 시점 정보들 중 어떤 정보를 Cell State에 전달할 지 결정

### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### 현 시점의 Cell State

: Forget Gate와 Input Gate의 출력을 바탕으로 현 시점의 Cell State 값 결정

$$C_t = f_t \times C_{t-1} + i_t \times g_t$$


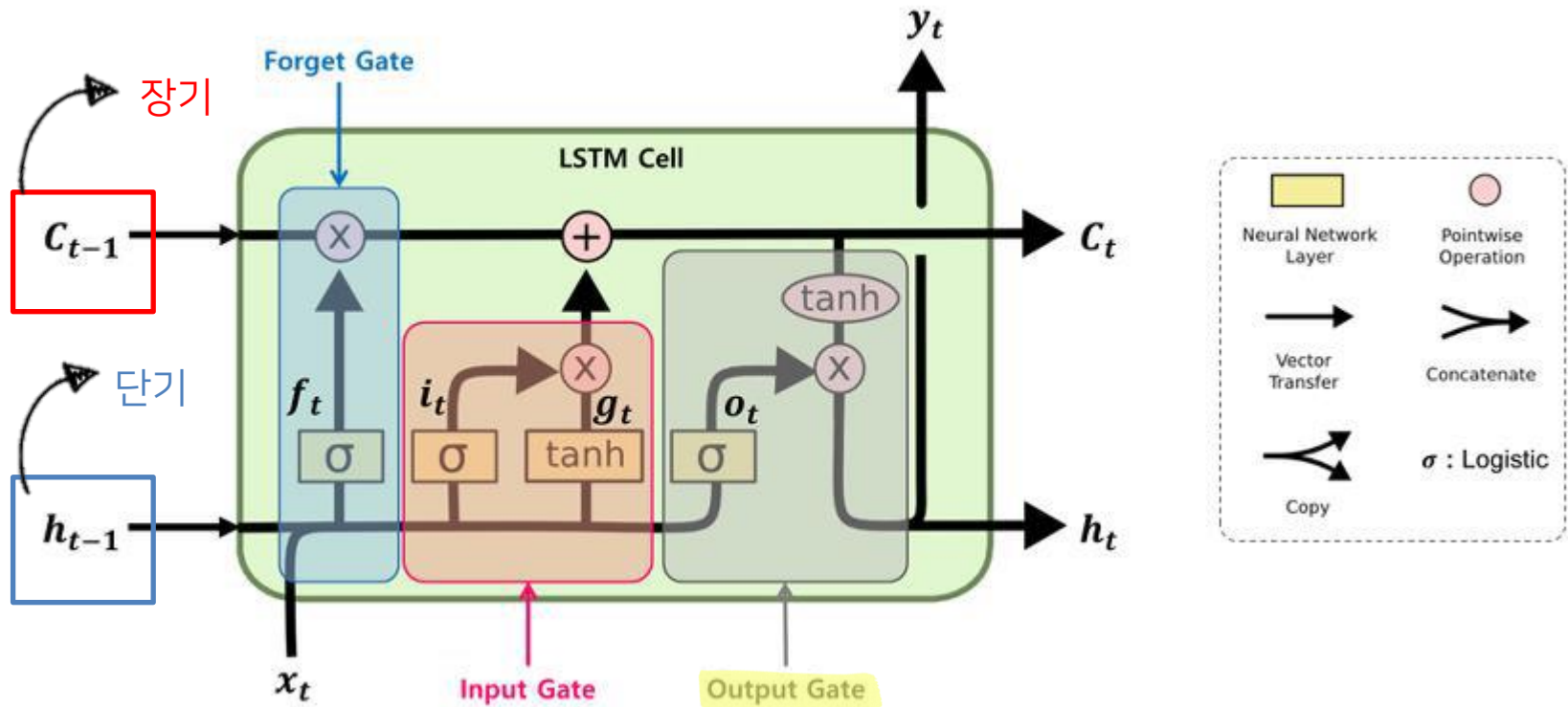
- Forget Gate : 얼마나 **이전 시점**의 정보들을 **잊을 것인지** 결정
- Input Gate : **현재 시점** 중 **어떤** 정보들을 **얼마나** 기억할지 결정

### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Output Gate

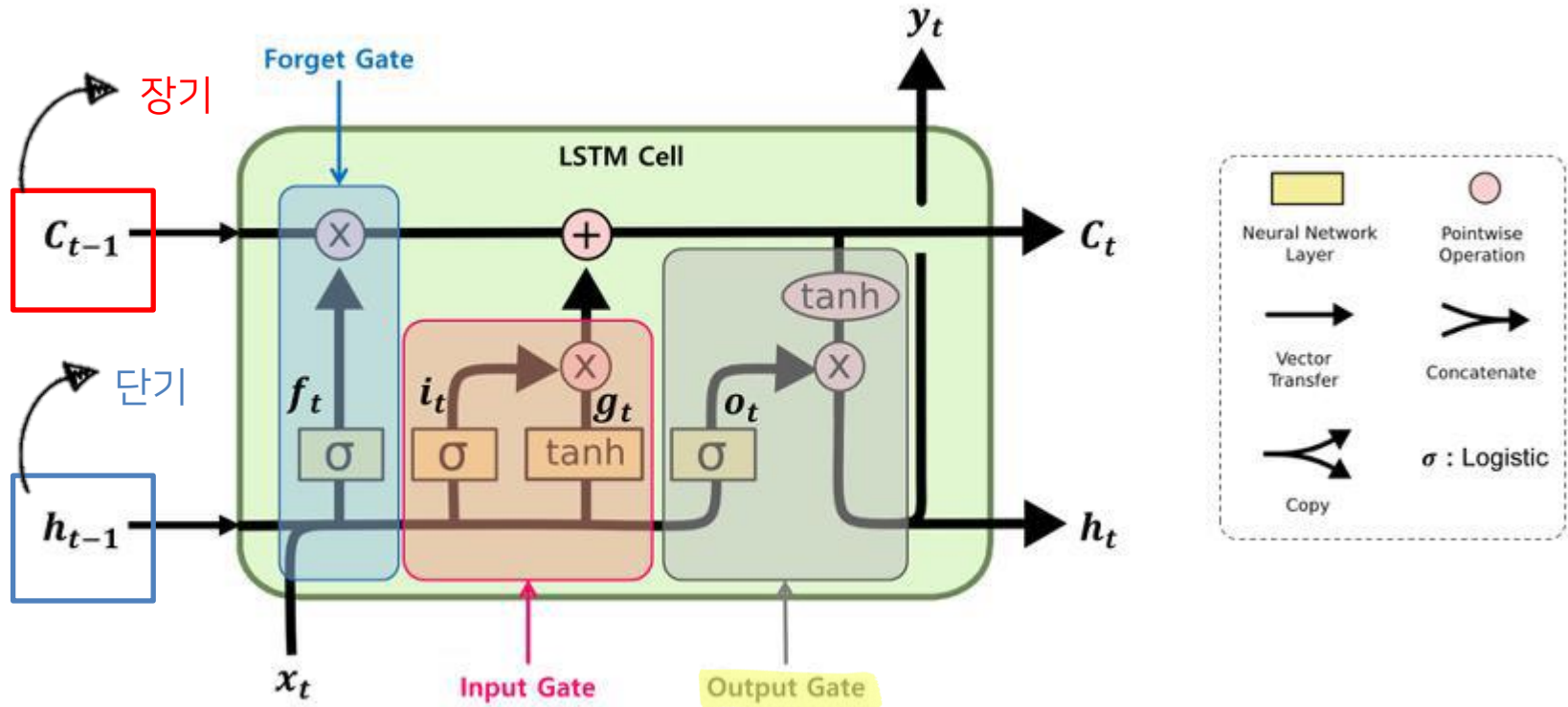
: 현재 시점의 Cell State와 이전 시점의 정보들을 바탕으로 현재 시점의 Hidden State 결정



### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Output Gate



현재까지의 정보들 중 어떤 정보를 얼마나 활용할 것인지 결정

### 3 RNN (Recurrent Neural Network)

- Hidden State 와 Cell State

#### Output Gate

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \times \tanh(C_t)$$

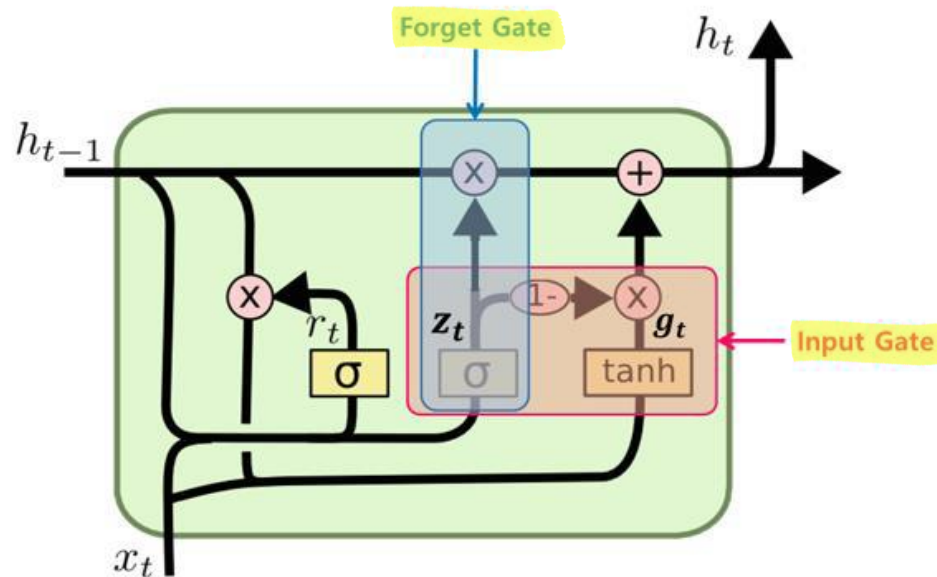
출력값은 현 시점의 **Output**이자 다음 시점이 전달 받을 **Hidden State**



### 3 RNN (Recurrent Neural Network)

- GRU

#### Output Gate



- LSTM의 원리와 유사함
- LSTM과는 달리 Cell State와 Output Gate가 없음
- Cell State를 제거하여 LSTM에 비해 파라미터 수가 적음

# 4

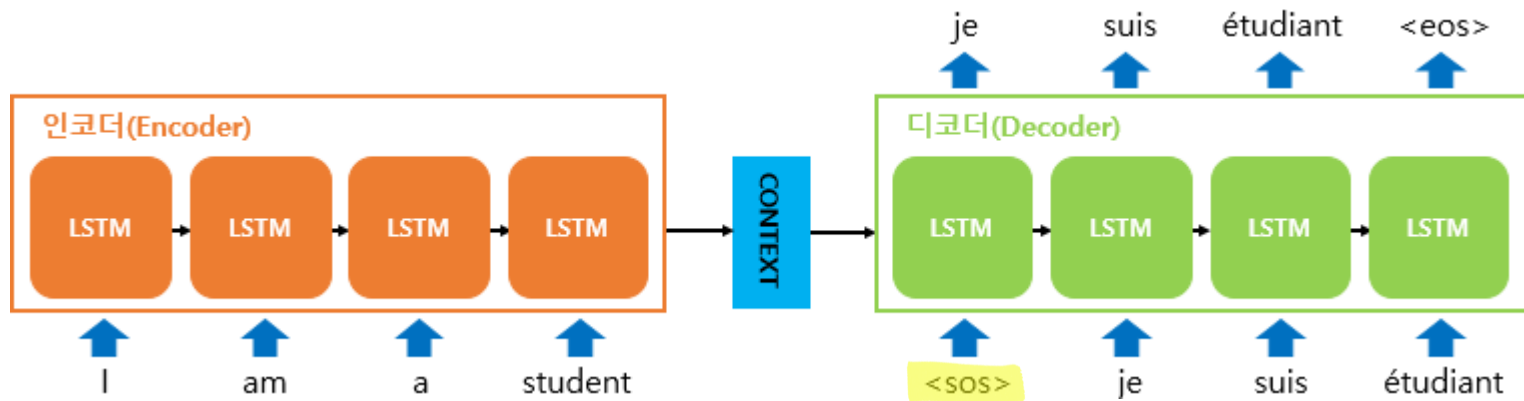
## RNN 모델의 응용

## 4 RNN 모델의 응용

- Seq2Seq

### Encoder-Decoder란?

- Encoder : Sequential Data를 입력받아 압축된 하나의 벡터로 만드는 역할
- Decoder : Encoder의 마지막 Hidden State와 입력을 바탕으로 Encoder와 달리 매 시점마다 출력을 내보냄

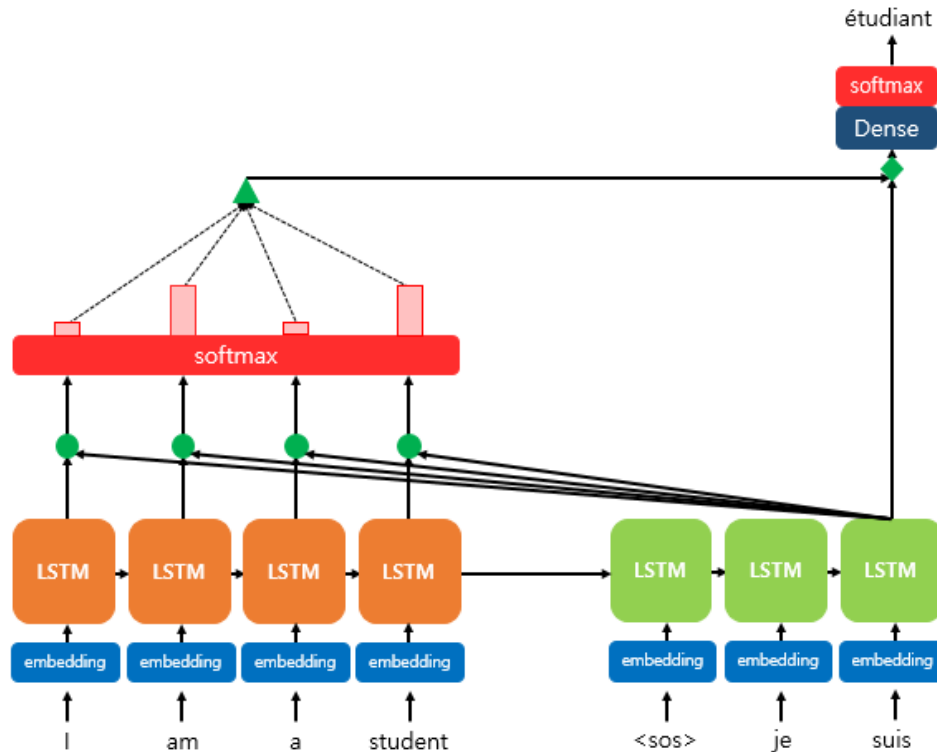


## 4 RNN 모델의 응용

- Attention

# Attention이란?

: Encoder-Decoder에서의 병목현상을 해결하기 위해 제안된 아이디어

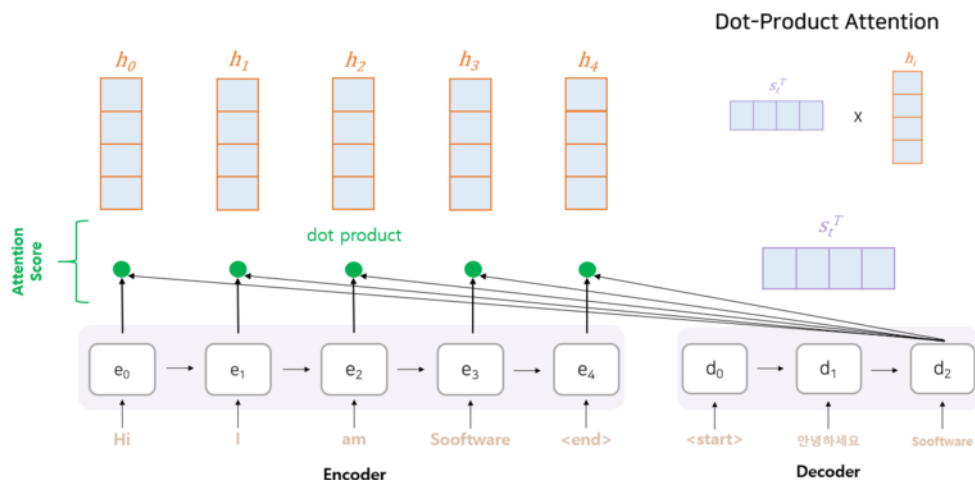


# 4 RNN 모델의 응용

## ● Attention

### Attention의 계산

- Decoder의 각 시점 Hidden State와 Encoder의 모든 시점 Hidden State의 유사도를 구함



$$score(s_t, h_i) = s_t^T h_i$$

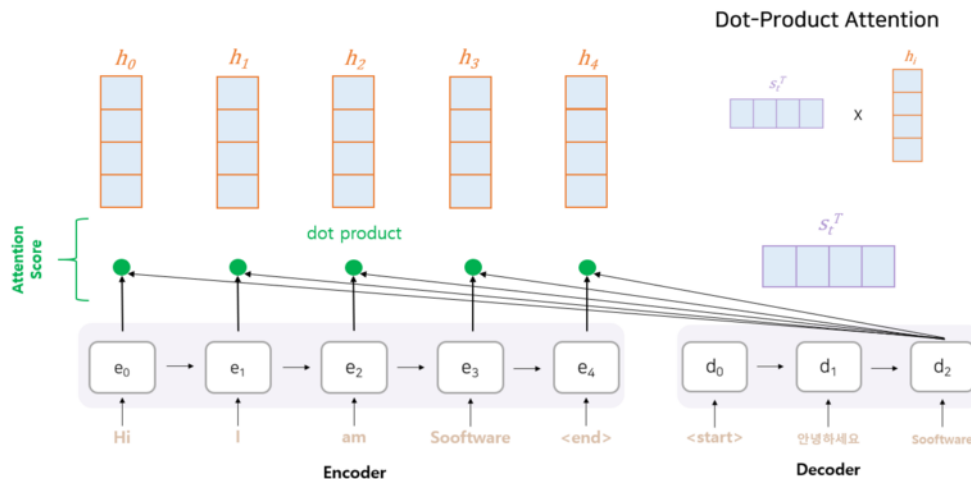
$$e^t = [s_t^T h_1, s_t^T h_2, \dots, s_t^T h_N]$$

# 4 RNN 모델의 응용

## ● Attention

### Attention Distribution 계산

- Attention Score 벡터를 Softmax 함수에 통과시켜 **확률값** 얻음



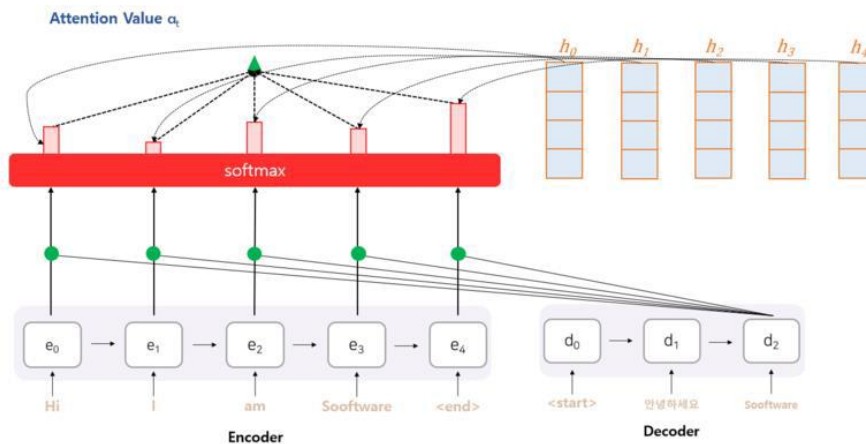
$$a^t = \text{softmax}(e^t)$$

# 4 RNN 모델의 응용

## ● Attention

### Attention Value 계산

- Attention Distribution을 Encoder의 모든 Hidden State와 곱함



$$\text{score}(s_t, h_i) = s_t^T h_i$$

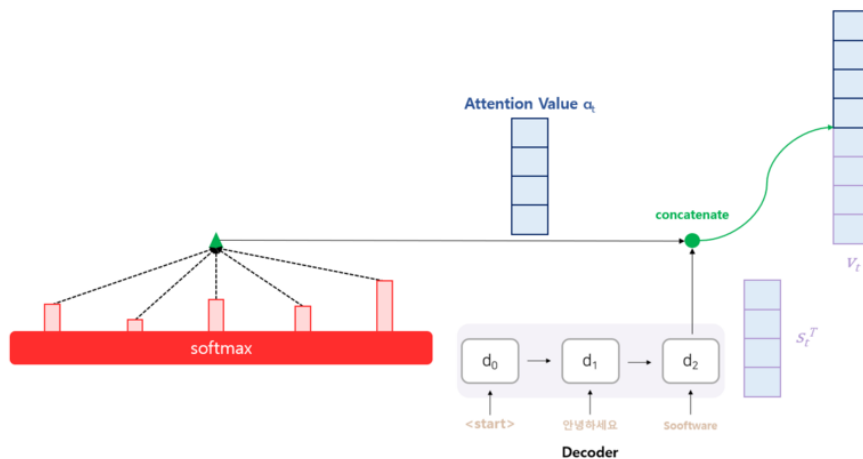
$$e^t = [s_t^T h_1, s_t^T h_2, \dots, s_t^T h_N]$$

## 4 RNN 모델의 응용

### ● Attention

#### Decoder의 Hidden State와 연결

- Hidden State와 Attention Value를 연결
- tanh 함수를 통과한 벡터가 출력층의 입력이 됨



$$\tilde{s}_t = \tanh(W_c \cdot v_t + b_c)$$

$$o_t = \text{softmax}(W_o \cdot \tilde{s}_t + b_o)$$





THANK YOU

