

## M3. 딥러닝 심화

## 더 나은 심층학습

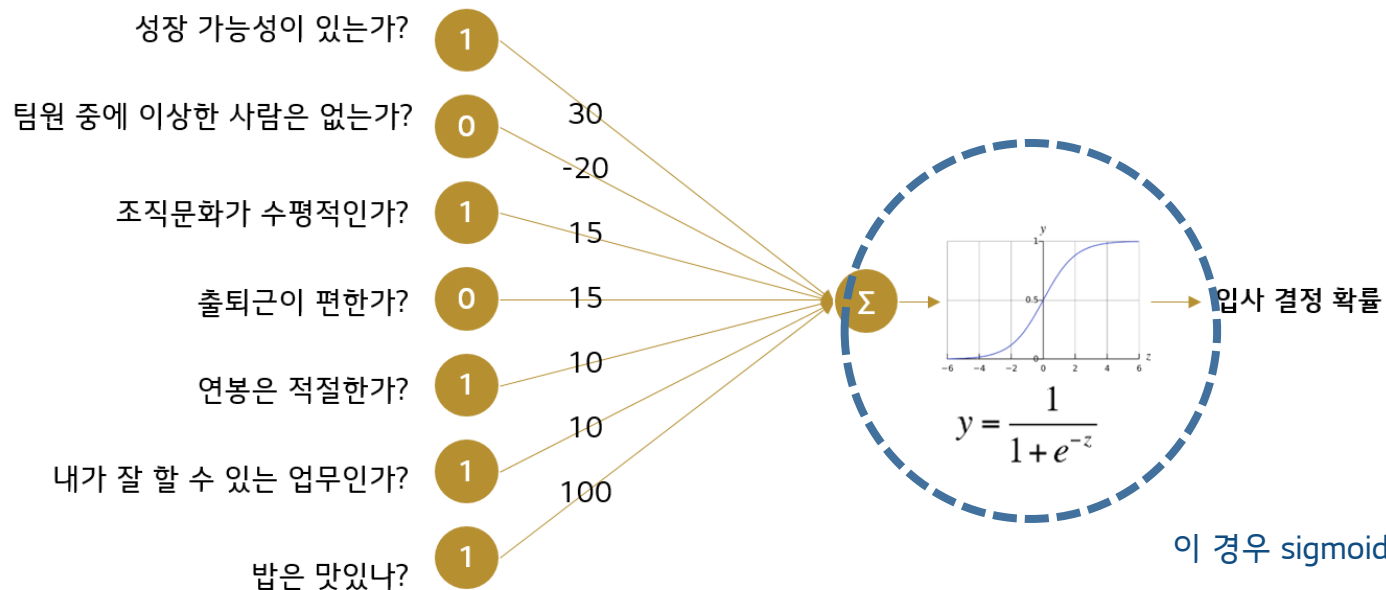
- Activation Function
- Loss function
- Optimizer
- Regularization
- CNN
- RNN

# Activation function

input과 weight의 가중합을 입력 받아 어떤 output을 내보낼지 결정하는 함수

cf. 2단원의 이 부분을 기억하십니까..!?



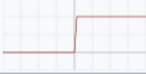
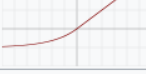


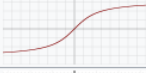

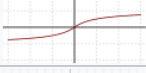
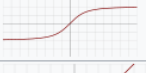





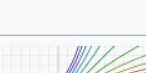

LG CNS에 입사할 것인가?



이 경우 sigmoid function이  
activation function으로서 채택!

# Activation function

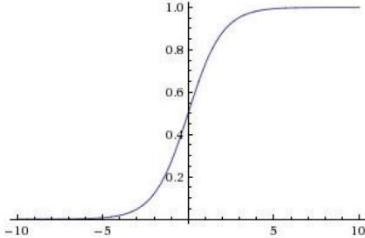
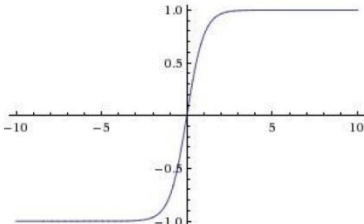
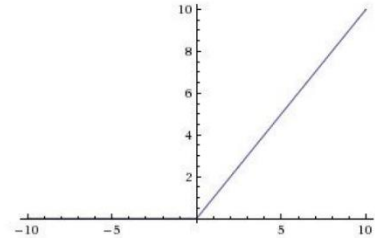
이제상엔 다양한 activation function이 있습니다..

Identity		$f(x) = x$	Randomized leaky rectified linear unit (RRReLU) <sup>[13]</sup>		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ <sup>[2]</sup>
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	Exponential linear unit (ELU) <sup>[14]</sup>		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$ <sup>[1]</sup>	Scaled exponential linear unit (SELU) <sup>[15]</sup>		$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	S-shaped rectified linear activation unit (SReLU) <sup>[16]</sup>		$f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ $t_l, a_l, t_r, a_r$ are parameters.
ArcTan		$f(x) = \tan^{-1}(x)$	Inverse square root linear unit (ISRLU) <sup>[9]</sup>		$f(x) = \begin{cases} \frac{x}{\sqrt{1 + \alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Softsign <sup>[7][8]</sup>		$f(x) = \frac{x}{1 +  x }$	Adaptive piecewise linear (APL) <sup>[17]</sup>		$f(x) = \max(0, x) + \sum_{s=1}^S \alpha_s^s \max(0, -x + b_s^s)$
Inverse square root unit (ISRU) <sup>[9]</sup>		$f(x) = \frac{x}{\sqrt{1 + \alpha x^2}}$	SoftPlus <sup>[18]</sup>		$f(x) = \ln(1 + e^x)$
Rectified linear unit (ReLU) <sup>[10]</sup>		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	Bent identity		$f(x) = \vee$
Leaky rectified linear unit (Leaky ReLU) <sup>[11]</sup>		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	Sigmoid-weighted linear unit (SiLU) <sup>[19]</sup> (a.k.a. Swish <sup>[20]</sup> )		$f(x) = x \cdot \sigma(x)$
Parametric rectified linear unit (PReLU) <sup>[12]</sup>		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	SoftExponential <sup>[21]</sup>		$f(\alpha, x) =$
Randomized leaky rectified linear unit (RRReLU) <sup>[13]</sup>		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ <sup>[2]</sup>			



# Activation function

자주 이용하는 activation function

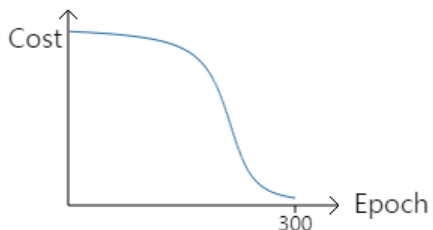
Sigmoid	Tanh	ReLU
$\sigma(x) = 1/(1 + e^{-x})$	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	*Rectified Linear Unit $\text{ReLU}(x) = \max(0, x)$
		
<ul style="list-style-type: none"><li>- 입력값을 0~1 사이 값으로 변환</li><li>- 전통적으로 많이 이용하는 방식</li></ul> <p>[문제]</p> <ul style="list-style-type: none"><li>- 몫시 크거나 작은 값을 입력으로 받는 경우, gradient가 거의 0에 가까워짐</li><li>- 계산이 다소 복잡</li></ul>	<ul style="list-style-type: none"><li>- 입력값을 -1~1 사이 값으로 변환</li></ul> <p>[문제]</p> <ul style="list-style-type: none"><li>- 역시, 몫시 크거나 작은 값을 입력으로 받는 경우 gradient가 거의 0에 가까워짐</li></ul>	<ul style="list-style-type: none"><li>- gradient가 0이 될 일 없음(+구역)</li><li>- 매우 간단한 계산</li><li>- 실제로 sigmoid나 tanh에 비해 빠르게 수렴하곤 함 (약 6배)</li></ul>

# Cost/Loss function

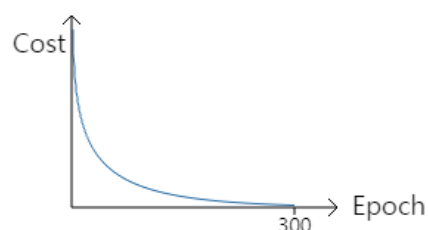
모델의 예측 결과와 실제 정답값의 차이를 정량화해주는 함수

Quadratic	Cross Entropy	Negative Log Likelihood
$C(w, b) = \frac{1}{2n} \sum_{i=1}^n (a_i - y_i)^2$ <ul style="list-style-type: none"><li>- 전통적인 방식의 loss 계산법</li><li>- (-) 하지만 sigmoid와 같이 이용할 경우 수렴을 더디게 함</li><li>- (+) 정답이 실수형인 경우에도 사용 가능 (예: regression)</li></ul>	$C(w, b) = -\frac{1}{n} \sum_{i=1}^n [y_i \log a_i + (1 - y_i) \log(1 - a_i)]$ <ul style="list-style-type: none"><li>- 이진분류에서 이용되는 함수</li><li>- (+) Quadratic Cost에 비해 모델 수렴을 빠르게 함</li><li>- 최근 DL framework에서 Multiclass NLL loss와 거의 동일하게 사용</li></ul>	$C(w, b) = -\sum_{i=1}^n \log a_{iy}$ <ul style="list-style-type: none"><li>- Multi-class 분류에서 이용되는 함수</li><li>- 정답에 해당하는 class만을 고려</li><li>- 이진분류의 경우 cross-entropy loss와 동일</li><li>- (+) Softmax 함수와 결합하여 사용할 때 모델 수렴을 빠르게 함</li></ul>

동일 모델에 다른 Cost function 적용해볼 때 수렴 속도 차이



Quadratic 적용



Cross entropy 적용

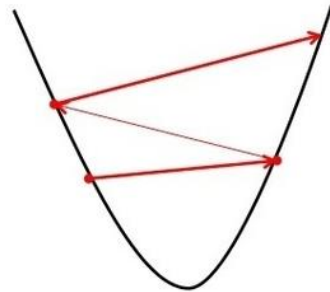
# Optimizer

Gradient 정보를 이용해 모델을 더 나은 방향으로 update하는 방법

2단원에서 배운 Gradient descent를 떠올려봅시다

$$w_{j+1} \leftarrow w_j - \alpha \frac{\partial C(w)}{\partial w_j}$$

Big learning rate



Small learning rate



Learning rate가 커도 문제, 작아도 문제,  
학습 내내 계속 같은 값이어도 문제..  
아 넘 설정하기 귀찮단말이죠...  
알아서 좀 할수 없을까요

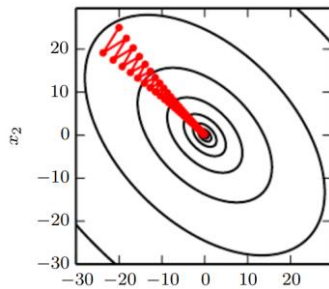


# Optimizer

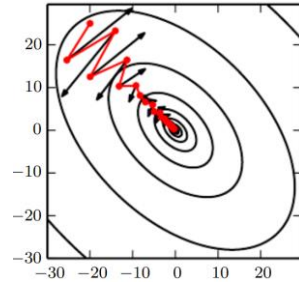
왜 없습니까! 다 방법이 있지요!!

## Momentum 계열 optimizer

과거의 파라미터 업데이트 내역을 누적,  
진행하던 방향성을 반영하여 빠르게 최적점으로 업데이트



SGD without momentum



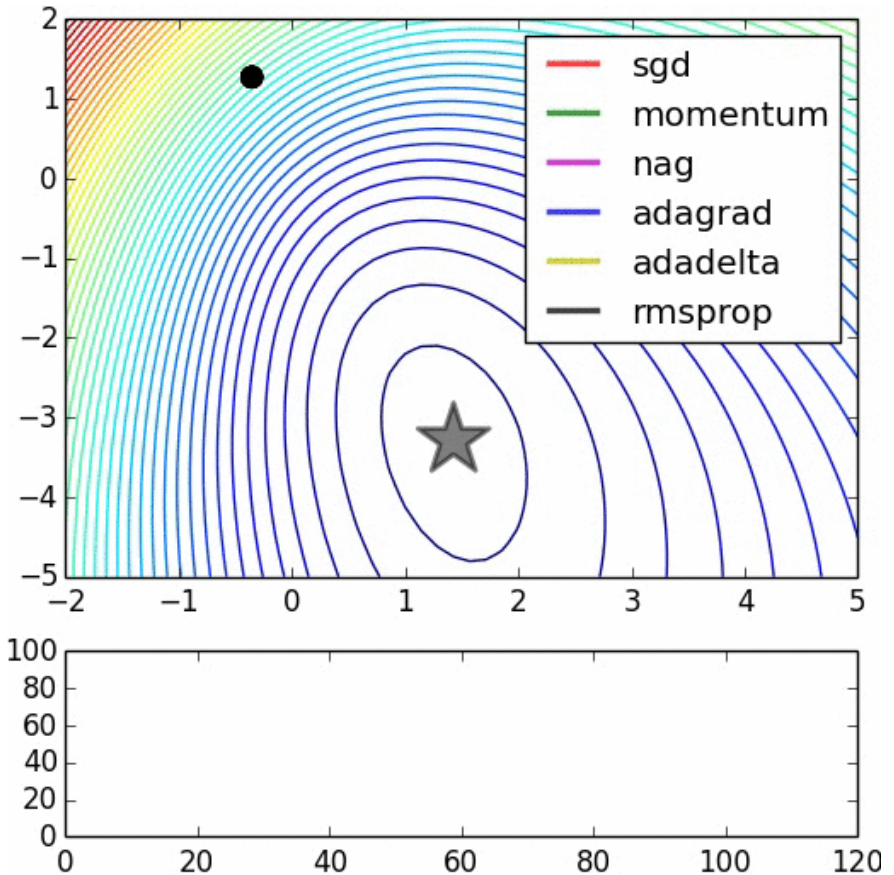
SGD with momentum

## Adaptive 계열 optimizer

파라미터 업데이트가 진행될수록 learning rate가 점점 줄  
어들어 미세하게 최적점을 찾아가도록 자동 조절

대표적 optimizer

- AdaGrad
- AdaDelta
- RMSprop
- AdaM





# Weight Initialization(가중치 초기화)

네트워크의 초기 weight 설정을 아무렇게나 할 경우 학습이 더디게 진행될 수 있다

- Weight 초기값을 어떻게 초기화 하느냐에 따라 학습이 잘 되거나 잘 안 되는 경우가 많다.
- Weight 초기값을 동일한 값으로 초기화하거나 모두 0으로 초기화 해서는 안 된다.
  - 가중치 초기값이 동일한 값일 경우 모든 뉴런이 동일한 출력 값을 내보내게 된다.
  - Backpropagation 단계에서 각 뉴런이 모두 동일한 gradient 값을 가지게 되고 결과적으로 뉴런의 개수가 아무리 많아도 뉴런이 하나 뿐인 것처럼 작동하게 되기 때문에 제대로 학습이 이루어지지 않는다.

## 효과적인 Weight Initialization 방법

※  $n_{in}$  : 이전 layer(input)의 노드 수,  $n_{out}$  : 다음 layer(input)의 노드 수

1. LeCun Normal Initialization : 가우시안 분포에서 분산을 X의 원래 분산 정도로 보정 (ReLU가 나오기 전)

$$W \sim N(0, Var(W))$$
$$Var(W) = \sqrt{\frac{1}{n_{in}}}$$

[비교] LeCun Uniform Initialization :  $W \sim U(-\sqrt{\frac{1}{n_{in}}}, +\sqrt{\frac{1}{n_{in}}})$

2. Xavier Initialization : 입력/출력 노드 수를 고려하여 초기값을 설정하는 방법 (ReLU가 나온 후)

$$W \sim N(0, Var(W))$$
$$Var(W) = \sqrt{\frac{2}{n_{in} + n_{out}}}$$

[비교] Xavier Uniform Initialization :  $W \sim U(-\sqrt{\frac{6}{n_{in} + n_{out}}}, +\sqrt{\frac{6}{n_{in} + n_{out}}})$

3. He initialization : Xavier Initialization 분산값을 2로 곱하는 방법 (활동함수가 ReLU 일 때 특히 유리하다.)

$$W \sim N(0, Var(W))$$
$$Var(W) = \sqrt{\frac{2}{n_{in}}}$$

[비교] He Uniform Initialization :  $W \sim U(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}})$

수식?!  
걱정마시라!

모든 효과적인 weight initialization 방법들은 이름만 알면 대부분의 딥러닝 프레임워크에서 쉽게 적용할 수 있도록 되어있다!

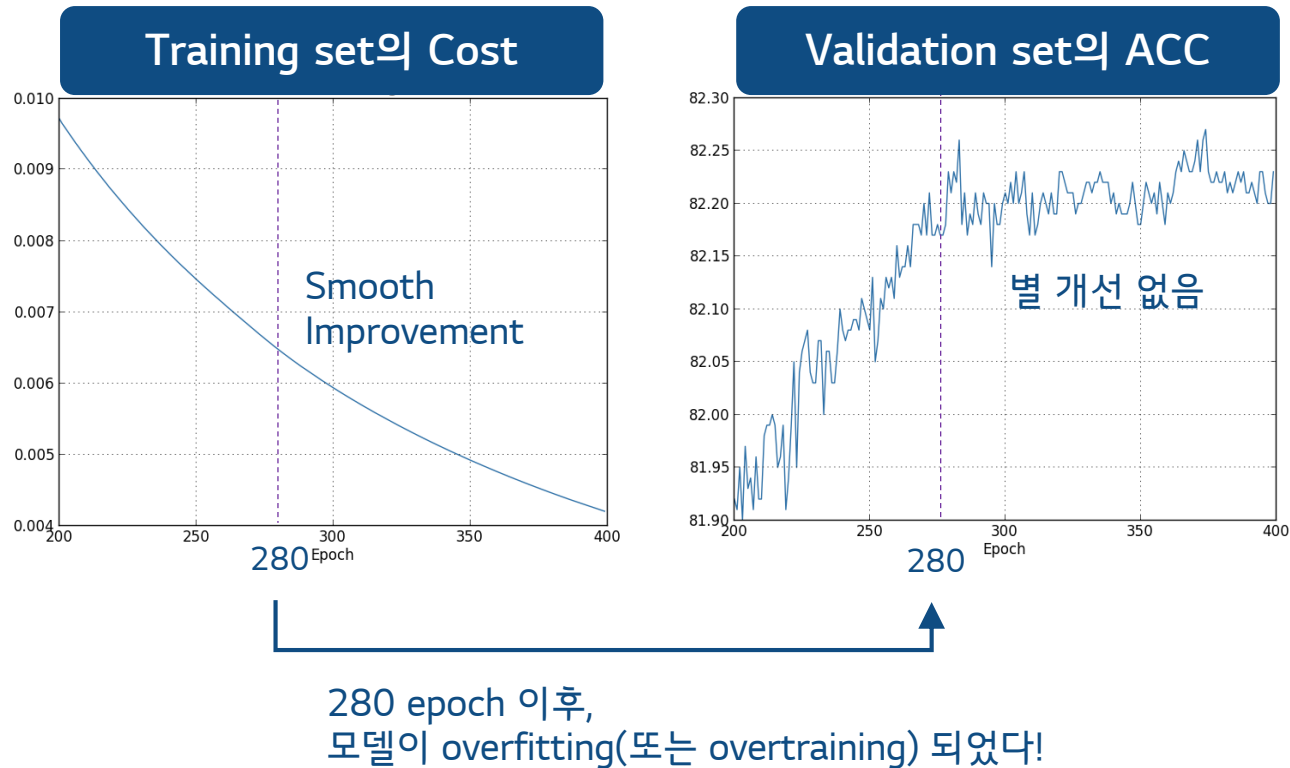
**Generalization error**를 감소시키려는 모든 노력

(※Training error 감소가 목적이 아님)

즉, **overfitting** 방지를 위함

# Regularization

Learning curve 해석하기 : MNIST 예제에서의 Overfitting

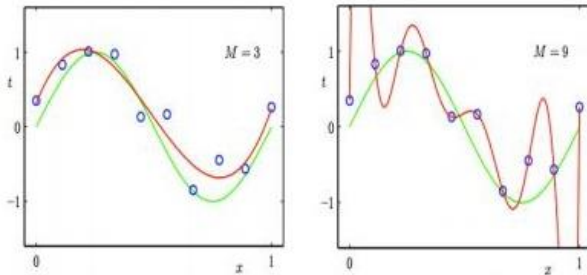


# Regularization

## Approach 1 : Model capacity 조절

Architecture 측면에서  
불필요하게 큰 모델 줄이기!

Hidden layer의 노드 수를 줄이거나...  
Layer 쌓는 수를 줄이거나 등등..



predictor too flexible:  
fits noise in the data

## Weight decay

아키텍처는 그대로,  
단 불필요한 weight를 0근처로 유도함으로써  
Capacity를 줄이는 것과 유사한 효과!

주 이용 방법 : Cost 뒤 L1 또는 L2 penalty를 부여

### L1 Regularization

$$C = C_0 + \frac{\lambda}{n} \sum_w |w|.$$

### L2 Regularization

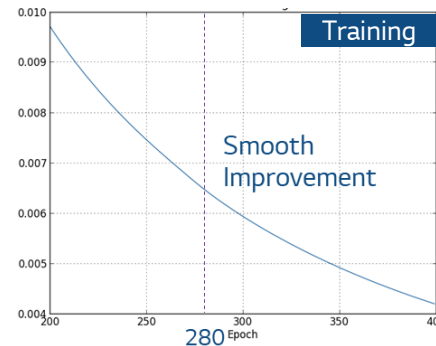
$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

$\lambda$  : Regularization 정도를 조절하는 hyperparameter

## Early stopping

Validation set의 Cost 및 성능을 모니터링,  
Overfitting의 조짐이 보이면 그 지점에서 학습 중단!

만일 우측과 같은 학습 양상이 보이는 경우,  
280 epoch까지만 학습된 모델을 이용



# Regularization

## Approach 2 : 더 많은 데이터 확보 (Data Augmentation)

### Image data Augmentation

좌우반전, Crop, 명암, 채도, 대비 변경 등..



### Text data Augmentation

(Image data에 비해 상대적으로 방법이 많지 않다)

Backtranslation 등..

The screenshot shows a web-based translation interface used for text augmentation via backtranslation. It consists of three stacked translation panels. The top panel is set to '한국어 - 감지됨' (Korean - Detected) and shows the Korean sentence '저는 밀떡볶이를 좋아해요' (I like wheat tteokbokki) being translated into English: 'jeoneun miltteogbokk-ileul joh-ahaeyo'. The middle panel is set to '영어 - 감지됨' (English - Detected) and shows the English sentence 'I like wheat tteokbokki' being translated back into Korean: 'I like wheat tteokbokki'. The bottom panel is set to '한국어' (Korean) and shows the final augmented Korean sentence '나는 밀 떡볶이를 좋아한다' (I like wheat tteokbokki). Blue curved arrows indicate the flow of the backtranslation process from the original Korean to English and then back to Korean. The interface includes language selection dropdowns, a search icon, and a star icon for saving.

# Regularization

## Approach 3 : 서로 다른 여러 모델 이용하기 (Ensemble)

머신러닝의 집단지성 : 일반적으로 여러 모델을 결합할수록 성능이 좋아진다!

### 다른 architecture

서로다른 아키텍처를 가진 모델 학습,  
학습된 모델의 추론 결과를 앙상블

### 다른 Hyperparameter

서로 다른 hyper parameter 세팅으로 학습한  
모델의 추론 결과를 앙상블

### 다른 training setp

한 모델을 학습하는 과정 중  
서로 다른 training step에 저장된 모델을 가져와  
추론 결과를 앙상블

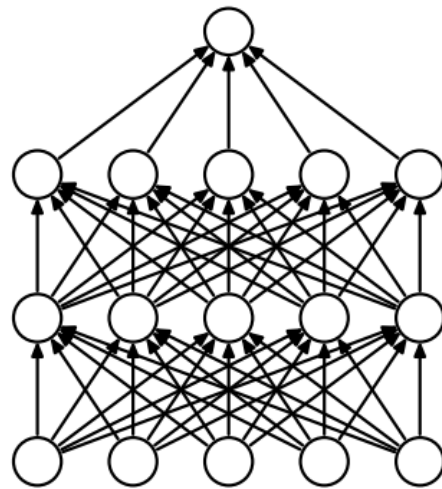
### 다른 initialization

동일한 아키텍처의 parameter를  
다른 방식으로 초기화하여 모델 학습,  
학습된 모델의 추론 결과 앙상블

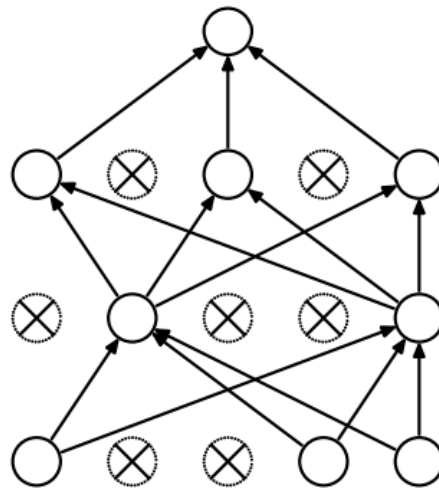
.. 이외 다양한 방법 가능

# Regularization

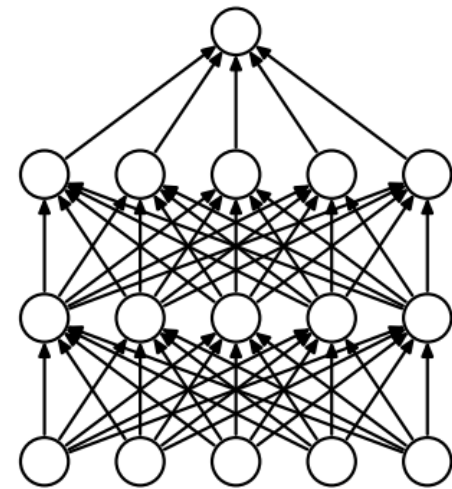
## Approach 4 : Dropout 이용하기



(a) Standard Neural Net



(b) After applying dropout.



(c) 학습이 끝나고 완성된 모델

Training

Inference

매 학습마다,  $p$ 의 확률로 각 뉴런을 drop할지 말지 결정  
Dropout된 뉴런은 일시적으로 삭제된 것으로 취급하고 학습

추론시 모든 뉴런을 살리되,  
결과스코어에  $p$ 만큼을 곱하여 이용

마치 매번 다른 네트워크를 학습시키는 것과 같은 효과

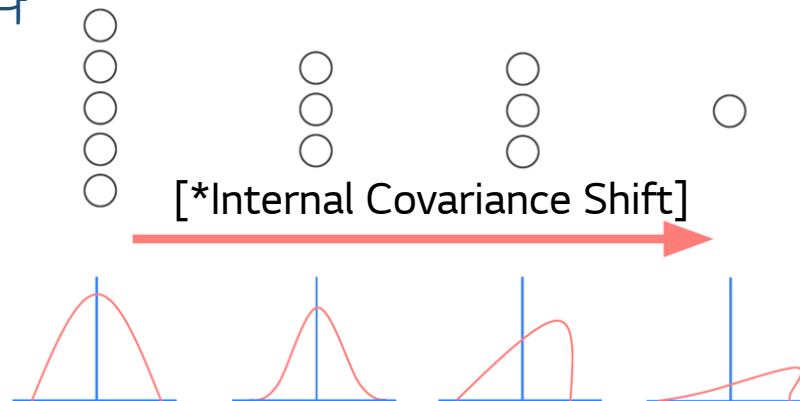
여러 네트워크를 앙상블하는 것과 같은 효과

# Regularization

## Approach 5 : Batch Normalization 이용하기

Gradient Vanishing / Gradient Exploding의 이유를 '\*Internal Covariance Shift'로 판단하여 이를 해결하는 아이디어 중의 하나

Input DATA



\*Internal Covariance Shift:  
Network의 각 층이나  
Activation 마다 input의  
distribution이 달라지는 현상

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots x_m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

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

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

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

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

[설명]

네트워크 학습시 mini-batch 단위로 데이터 학습,

→ Layer마다 mini-batch의 feature가 output으로 계산,

→ Feature의 평균과 표준편차를 구하여 normalize 해주고,

→ scale factor와 shift factor를 이용하여 새로운 값을 만들어준다.

보통 Batch Normalization 적용시 특정 Hidden Layer직전에 Batch Normalization Layer를 추가, input을 변경한 뒤 activation function에 넣어주는 방식으로 사용한다.

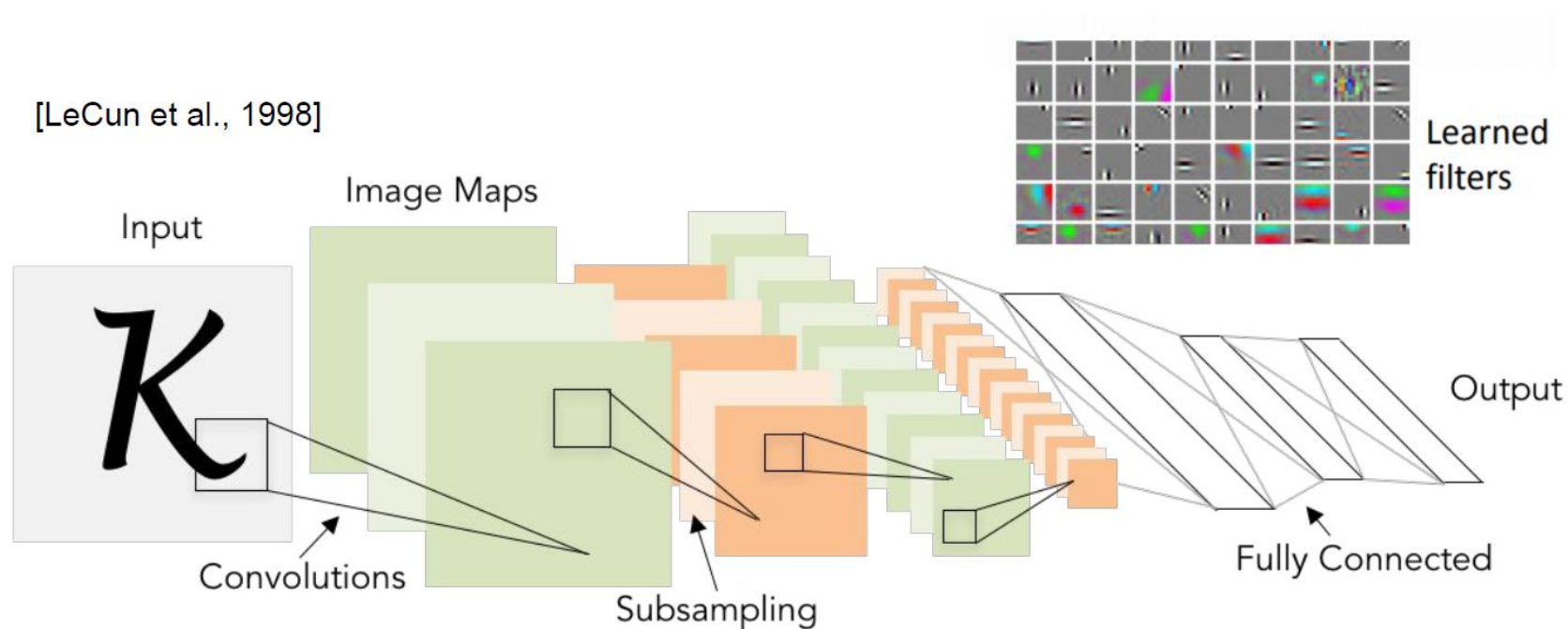


# Convolutional Neural Networks

## Convolutional Neural Networks 정의

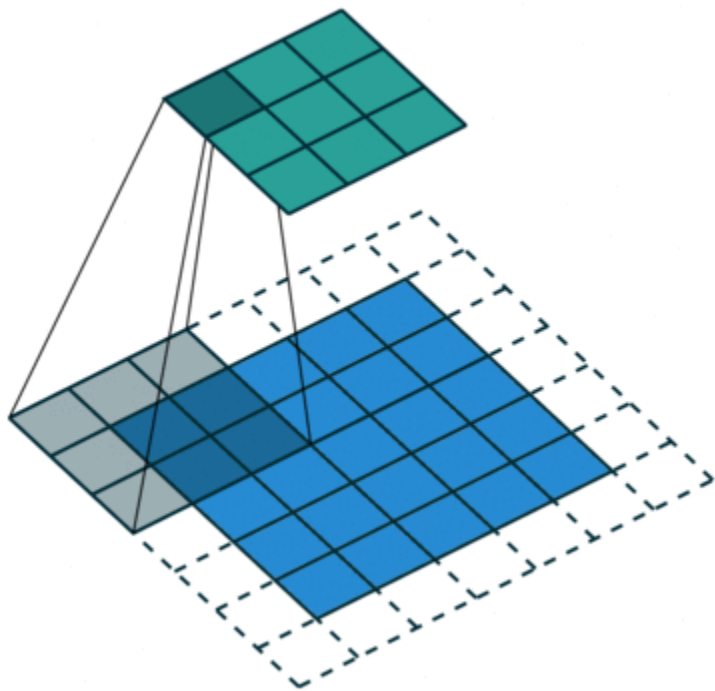
### Convolution + Subsampling(Pooling) + Full Connection

[LeCun et al., 1998]



# Convolutional Neural Networks

이미지로부터 특징을 추출하는 Convolution 연산



1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

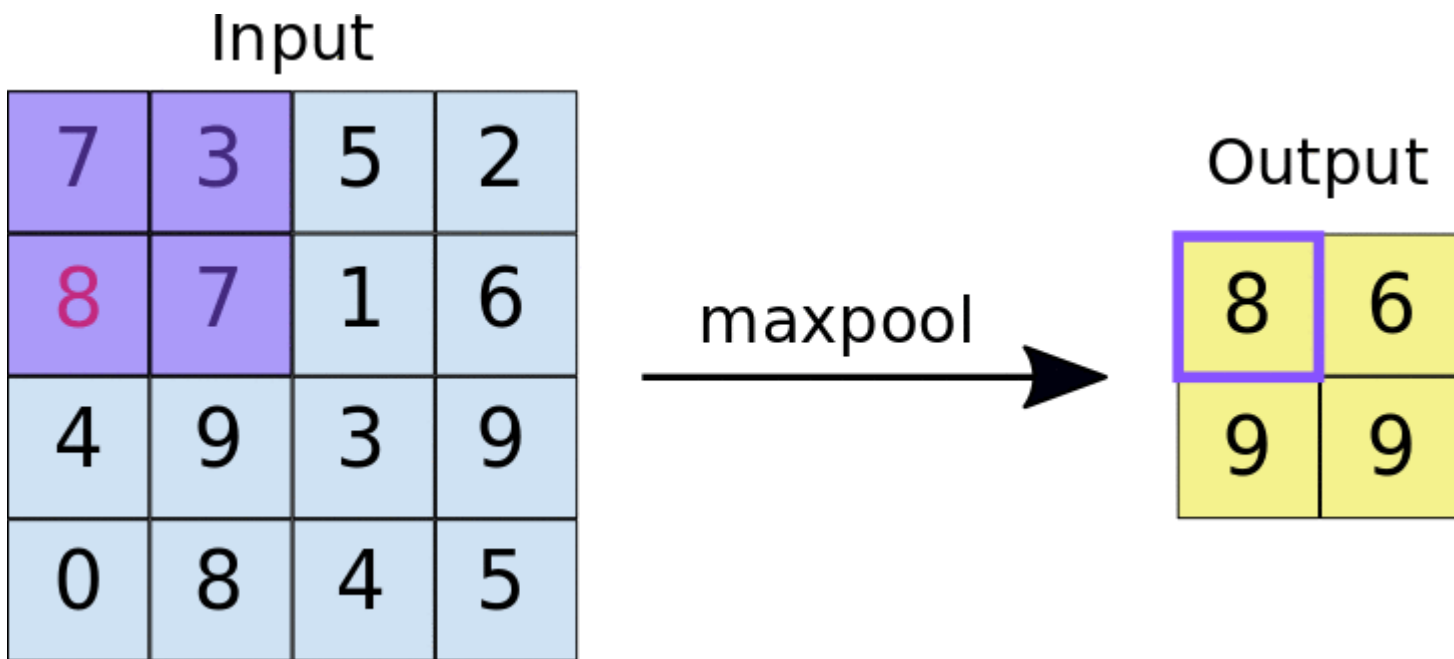
Image

4		

Convolved  
Feature

# Convolutional Neural Networks

차원을 축소하는 Subsampling(Pooling)

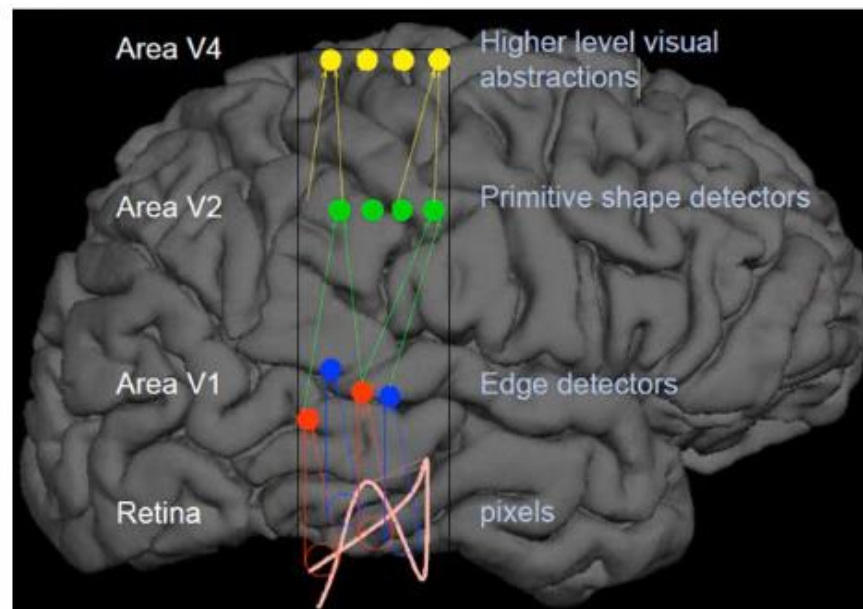


# Convolutional Neural Networks

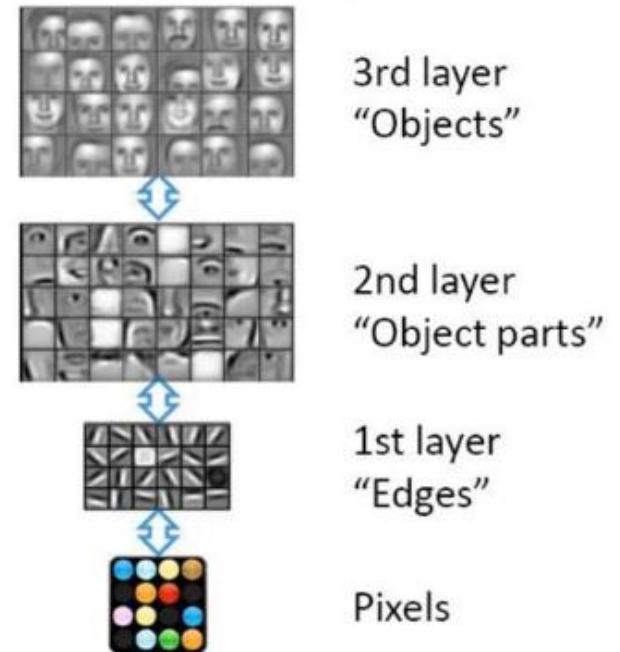
## Convolutional Neural Networks (a.k.a CNN, ConvNet)

### CNN을 쓰는 이유?

- 생물학적인 방법에서 고안
- 계층적으로 표현(representations)을 학습 (features)



### Feature representation



[Lee, Grosse, Ranganath & Ng, 2009]

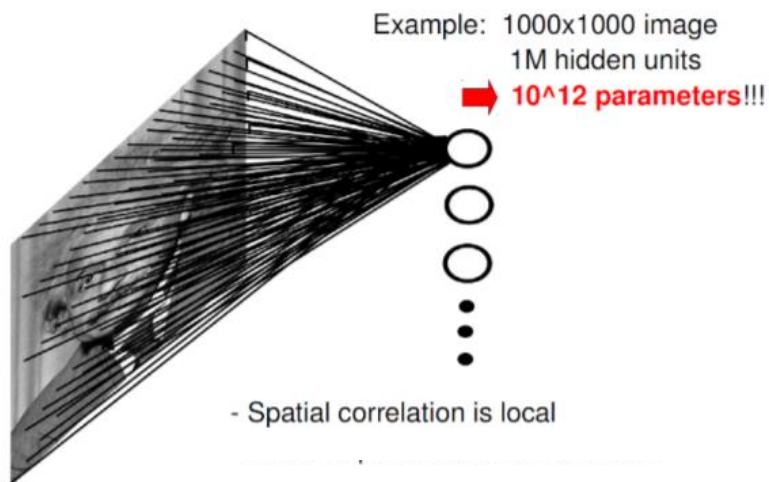
# Convolutional Neural Networks

## Convolutional Neural Networks (a.k.a CNN, ConvNet)

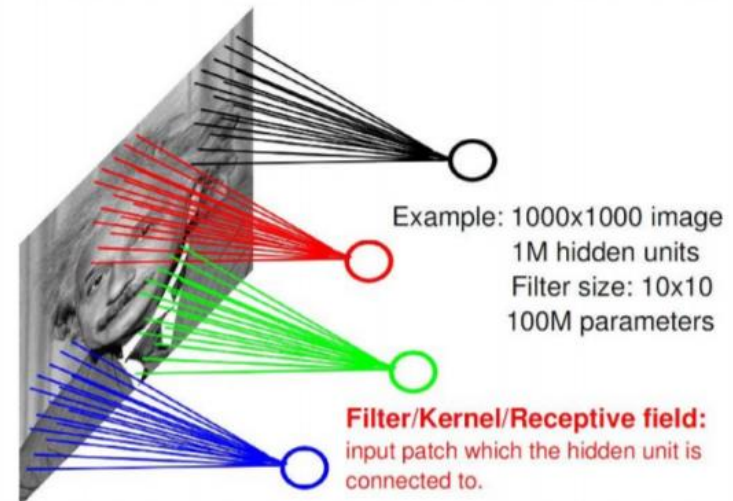
### CNN을 쓰는 이유?

- 부분적으로 인식(Receptive field)하는 것으로 공간구조를 유지
- Convolution 연산은 이미지의 공간적인 부분 상관관계 특성을 이용
  - 사람의 눈이 인식하는 것처럼 특정 위치와 해당 주변부에 집중

#### FULLY CONNECTED NEURAL NET



#### LOCALLY CONNECTED NEURAL NET

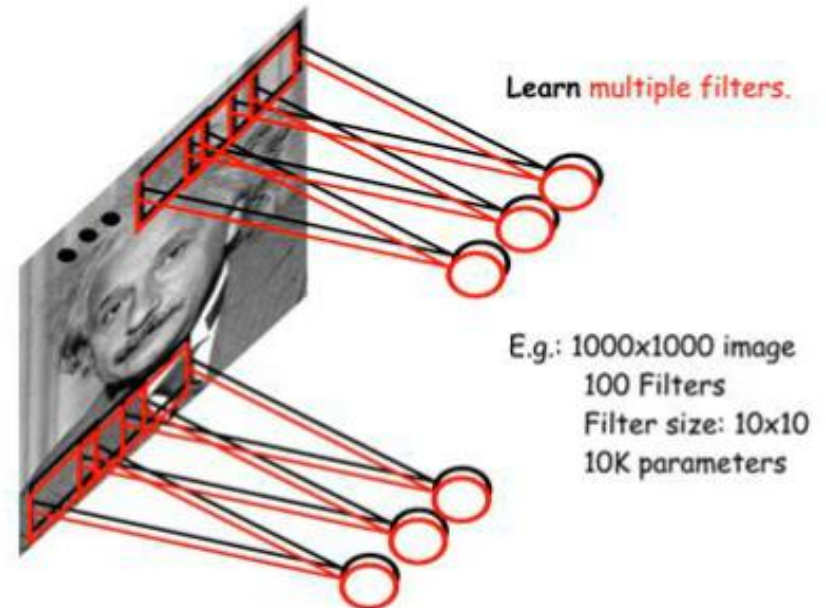
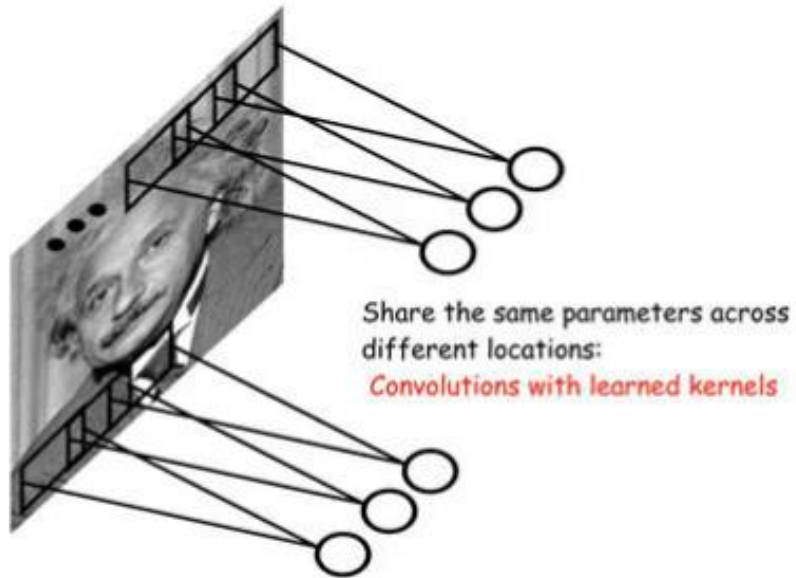


# Convolutional Neural Networks

## Convolutional Neural Networks (a.k.a CNN, ConvNet)

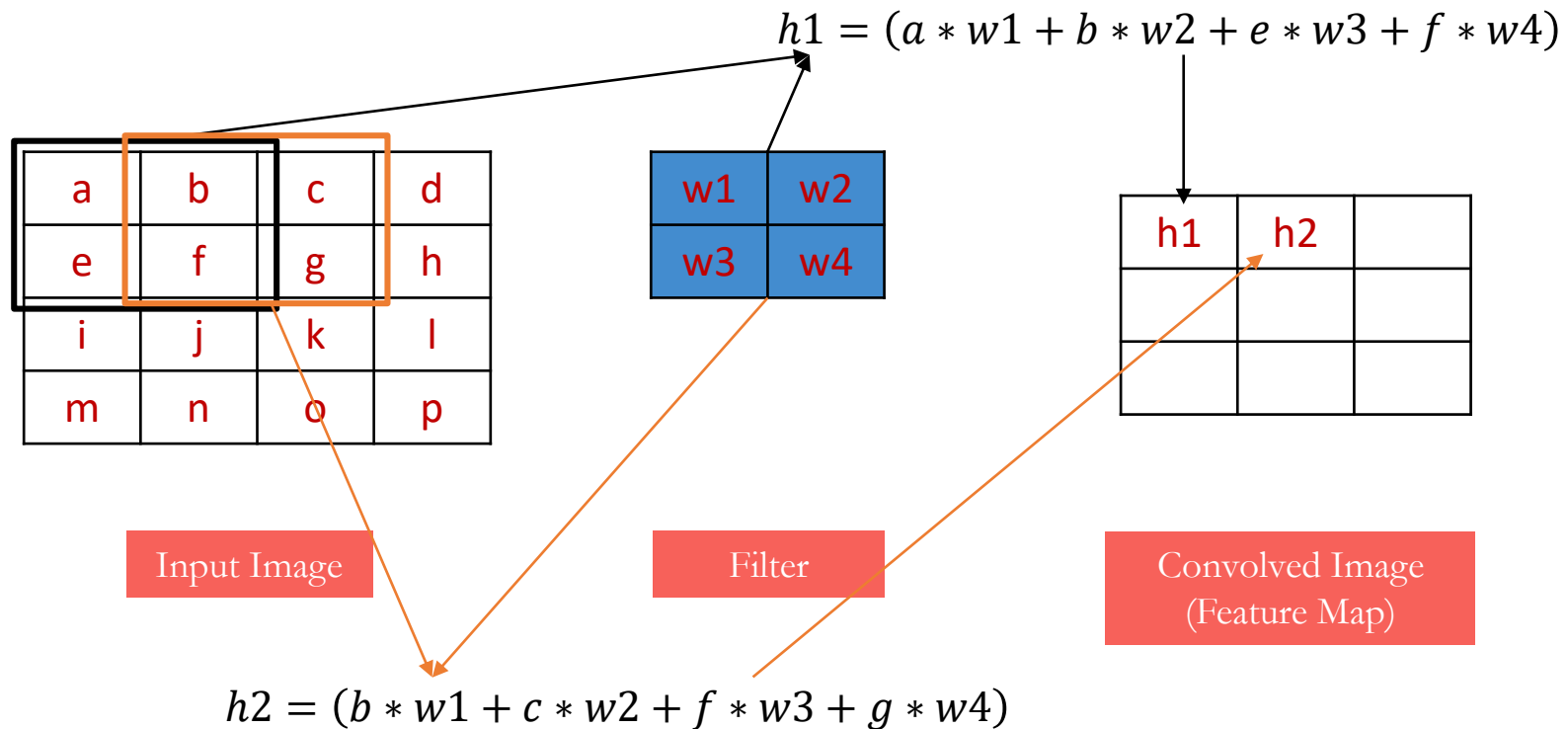
### CNN을 쓰는 이유?

- Weight를 공유
- FCN에 비하여 parameter 수가 크게 감소하여, overfitting을 줄여줌



# Convolutional Neural Networks

Convolution 연산은 적은 수의 parameter 만을 필요로 한다

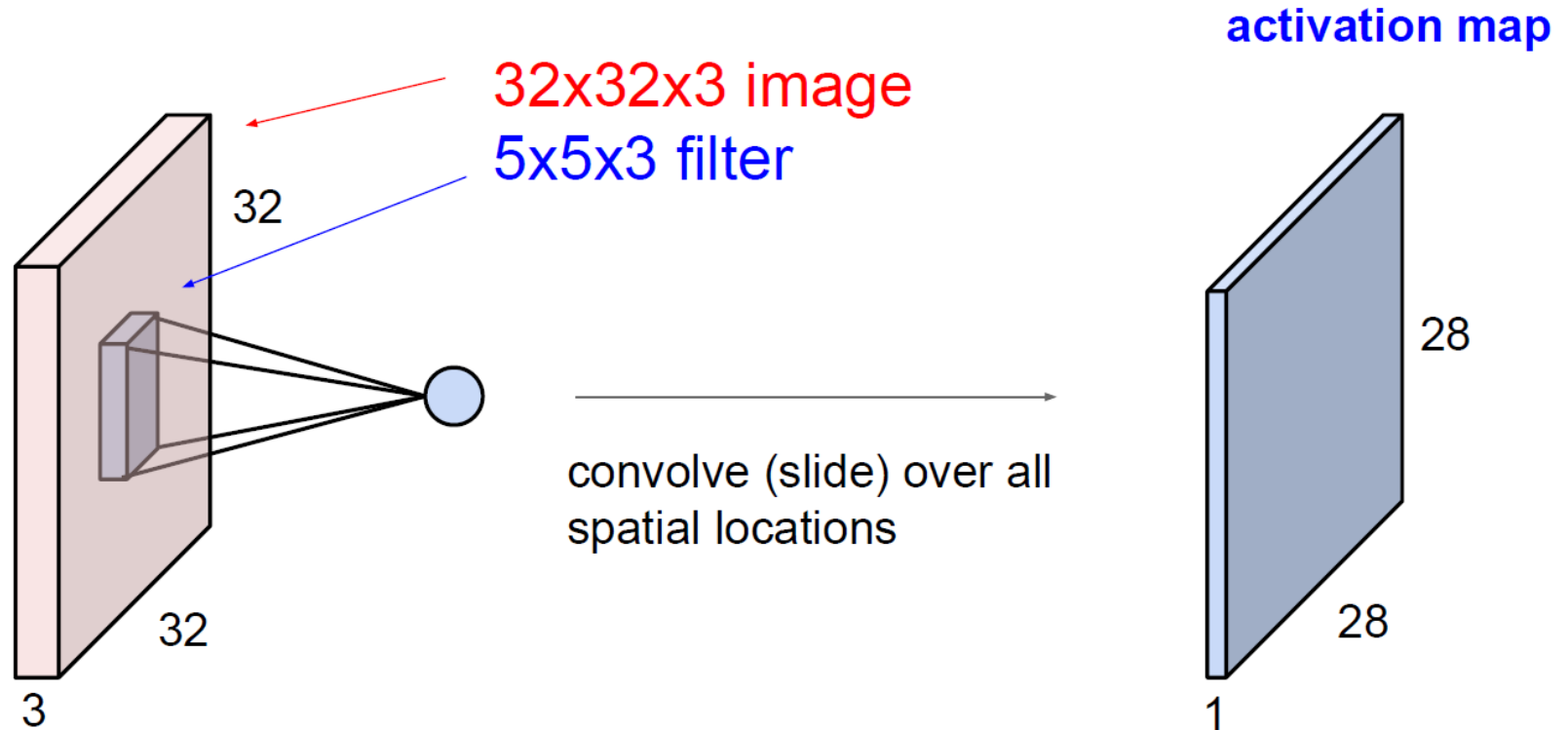


Number of Parameters for one feature map = 4

Number of Parameters for 100 feature map =  $4 * 100$

# Convolutional Neural Networks

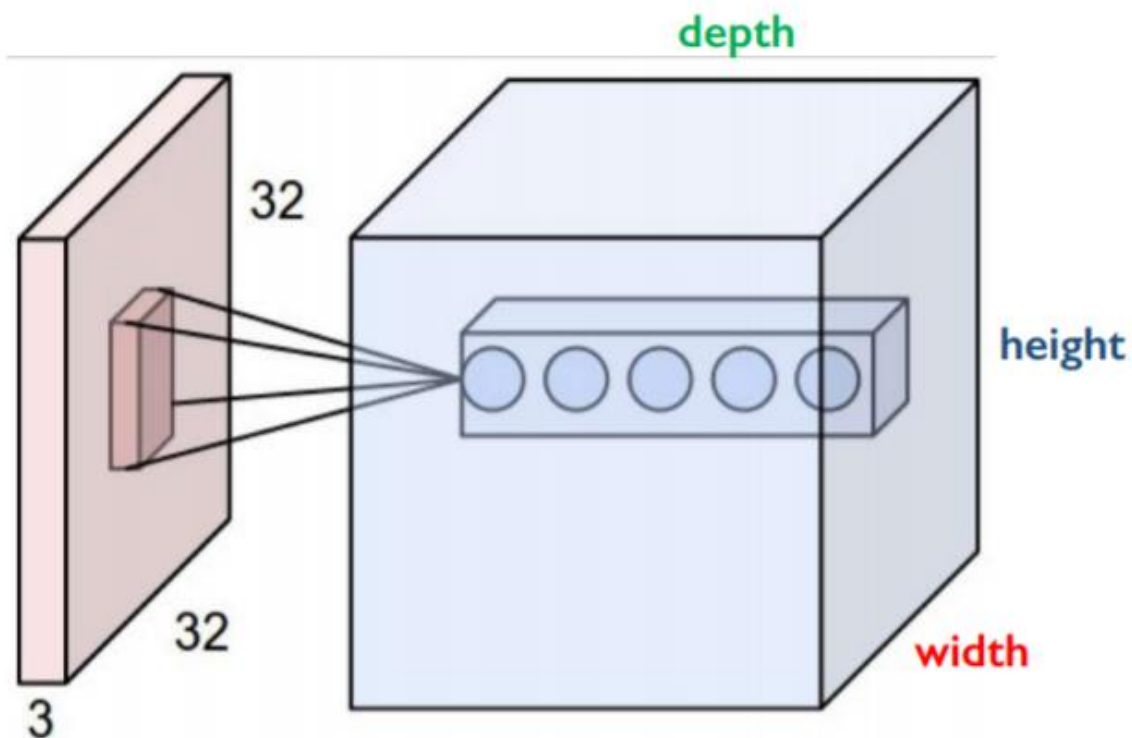
Convolution을 통해 feature를 추출





# Convolutional Neural Networks

feature를 여러 장 추출하면 이미지로부터 풍부한 정보를 꺼낼 수 있다



## Layer Dimensions:

$$h \times w \times d$$

where  $h$  and  $w$  are spatial dimensions  
 $d$  (depth) = number of filters

## Stride:

Filter step size

## Receptive Field:

Locations in input image that  
a node is path connected to

# Convolutional Neural Networks

Padding : 이미지 주변에 계산과는 무관한 테두리를 덧붙여 output의 사이즈를 조정

In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

**3x3** filter, applied with **stride 1**

**pad with 1 pixel** border => what is the output?

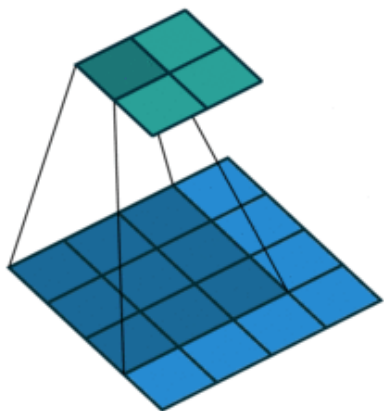
(recall:)

$$(N - F) / \text{stride} + 1$$

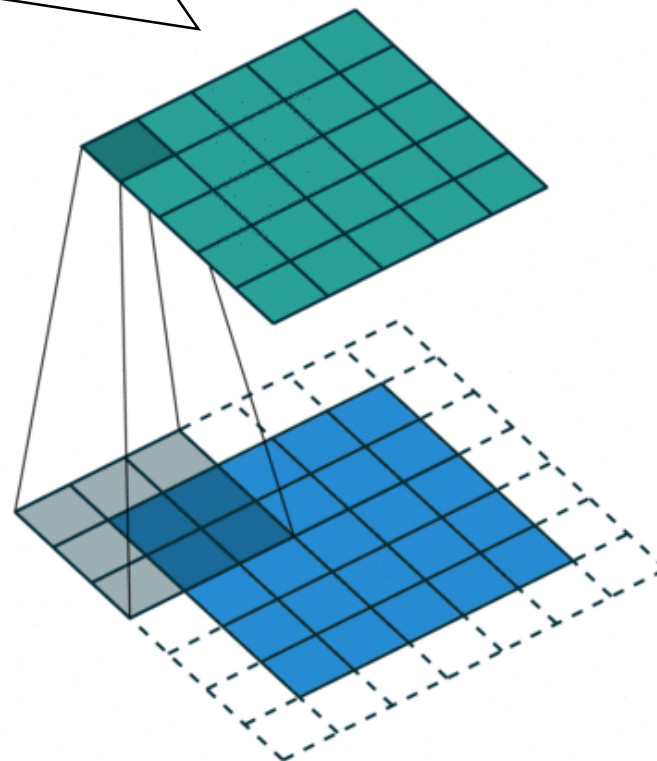
# Convolutional Neural Networks

Padding : 이미지 주변에 계산과는 무관한 테두리를 덧붙여 output의 사이즈를 조정

3x3 필터 적용시 output

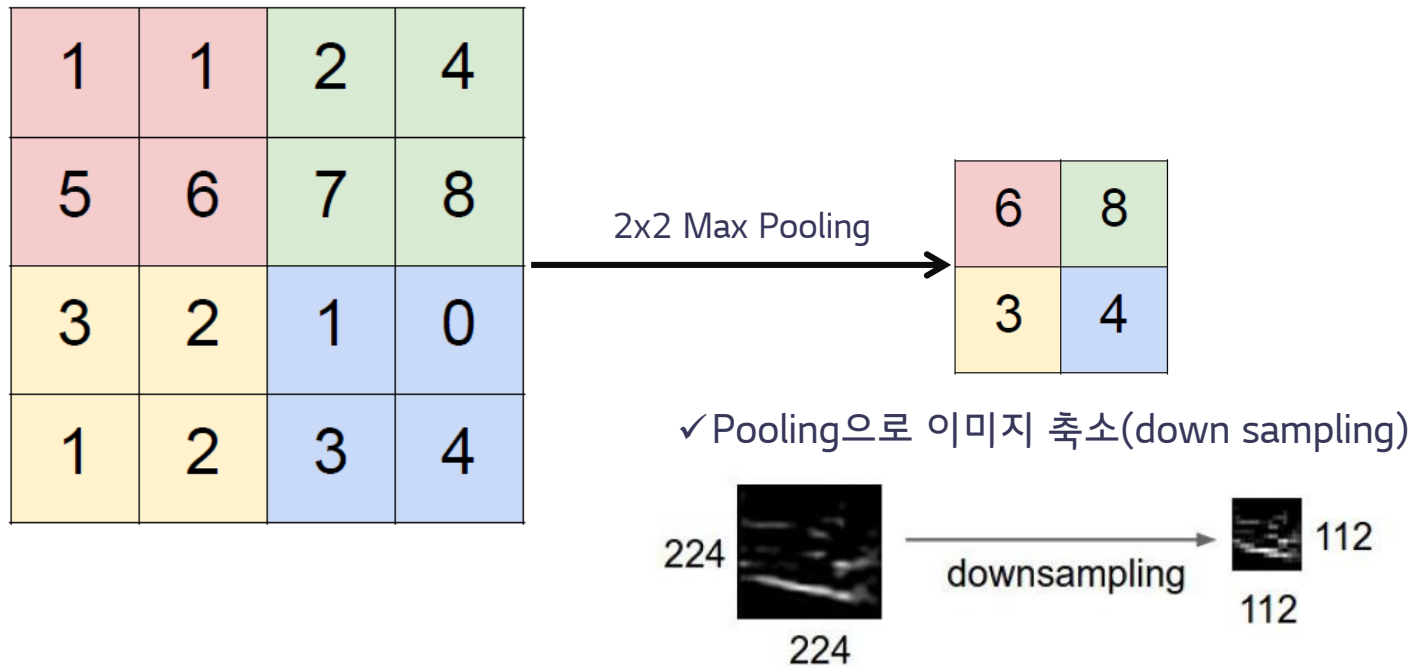


Padding 후 3x3 필터 적용시 output



# Convolutional Neural Networks

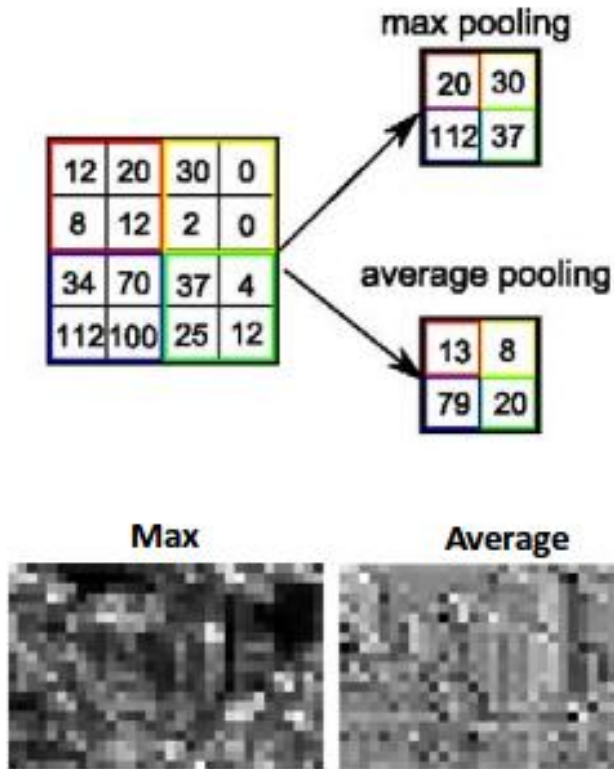
## Pooling 연산 : Max Pooling



- Pooling 연산: Filter를 이용하여 결과들을 합침
  - Conv layer의 출력에서 정보를 단순화
  - 차원을 줄임
  - 공간적인 의미로는 유지

# Convolutional Neural Networks

Pooling 연산 : Max Pooling, Average Pooling



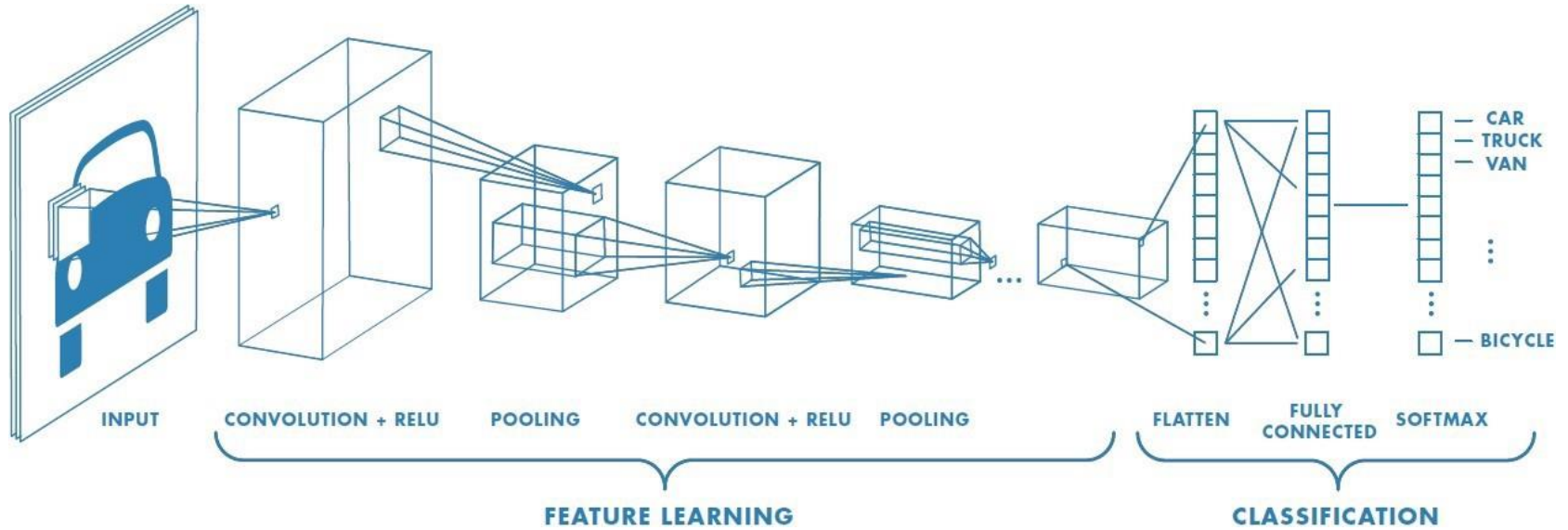
Produces a volume of size  $W_2 \times H_2 \times D_2$  where:

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$
- $D_2 = D_1$

- Max pooling 연산은 edge와 같은 가장 중요한 feature 들을 추출하는데 반해,
- Average pooling 연산은 비교적 smooth한 feature를 추출한다.

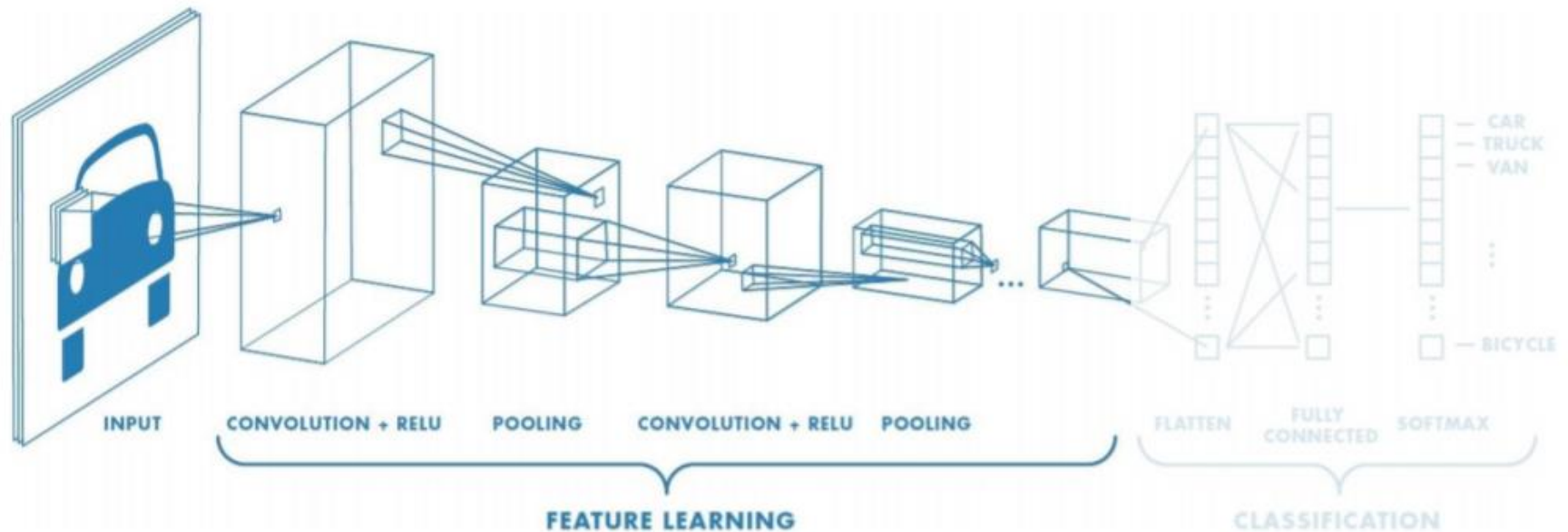
# Convolutional Neural Networks

## CNN for Classification



# Convolutional Neural Networks

## Feature Learning



1. 이미지로부터 convolution 연산을 통해 특징을 학습
2. Activation function을 거쳐 non-linearity 적용
3. Pooling으로 공간정보를 유지하면서도 차원을 축소

# Convolutional Neural Networks

## CNN for Classification



- Convolution과 Pooling으로 도출된 고차원의 featur를
- 분류를 하기 위해 납작하게 1차원으로 펼쳐서
- Class 수만큼 분류되도록 fully-connected로 연결!
- Class 수만큼의 probability로 표현

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

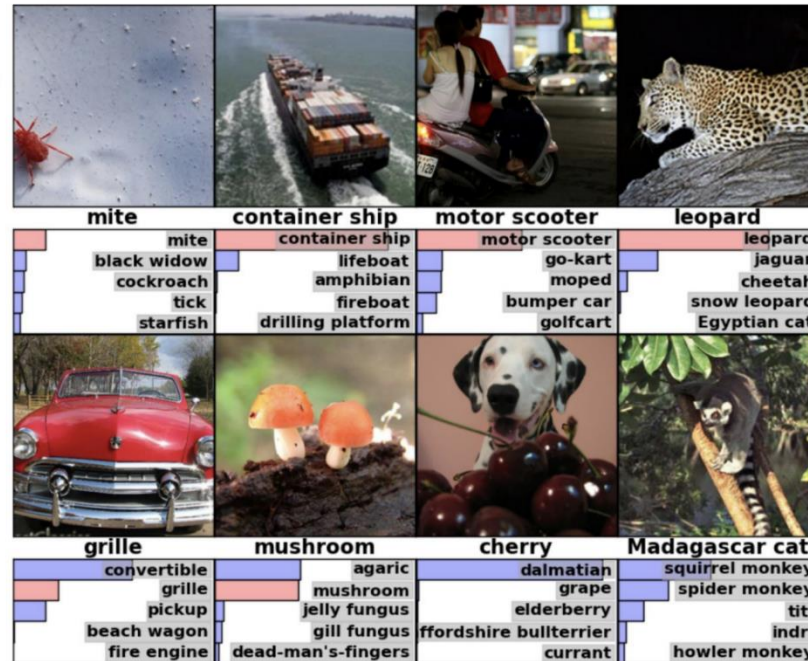


# Convolutional Neural Networks

ImageNet Challenge : 글로벌 이미지 분류 대회

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.



**Classification task:** produce a list of object categories present in image. 1000 categories.  
“Top 5 error”: rate at which the model does not output correct label in top 5 predictions

Other tasks include:

single-object localization, object detection from video/image, scene classification, scene parsing

# Convolutional Neural Networks

ImageNet 챌린지에서 우승한 주요 CNN 아키텍처

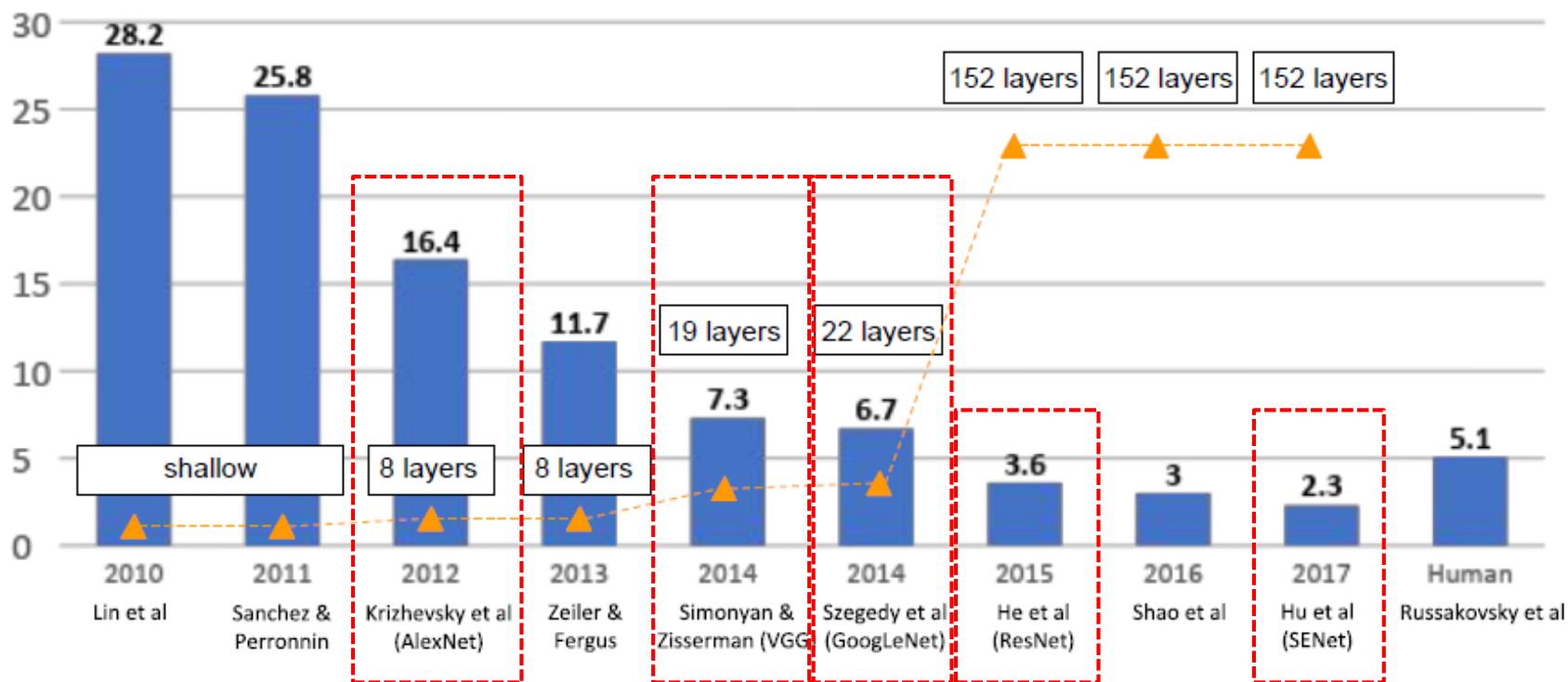
AlexNet

VGGNet

GoogLeNet

ResNet

DenseNet



# Convolutional Neural Networks

AlexNet : ImageNet 대회 첫 딥러닝(CNN기반) 모델 적용, 오류를 크게 개선

2012 ILSVRC Top-5 accuracy : 84.7% (2<sup>nd</sup> place is with 73.8%)

## 특징

- 처음으로 ReLU 적용
- Dropout 적용
- 2개의 path로 나누어 처리, 중간과정에 cross 공유 (자원제약)

## Architecture:

CONV1

MAX POOL1

NORM1

CONV2

MAX POOL2

NORM2

CONV3

CONV4

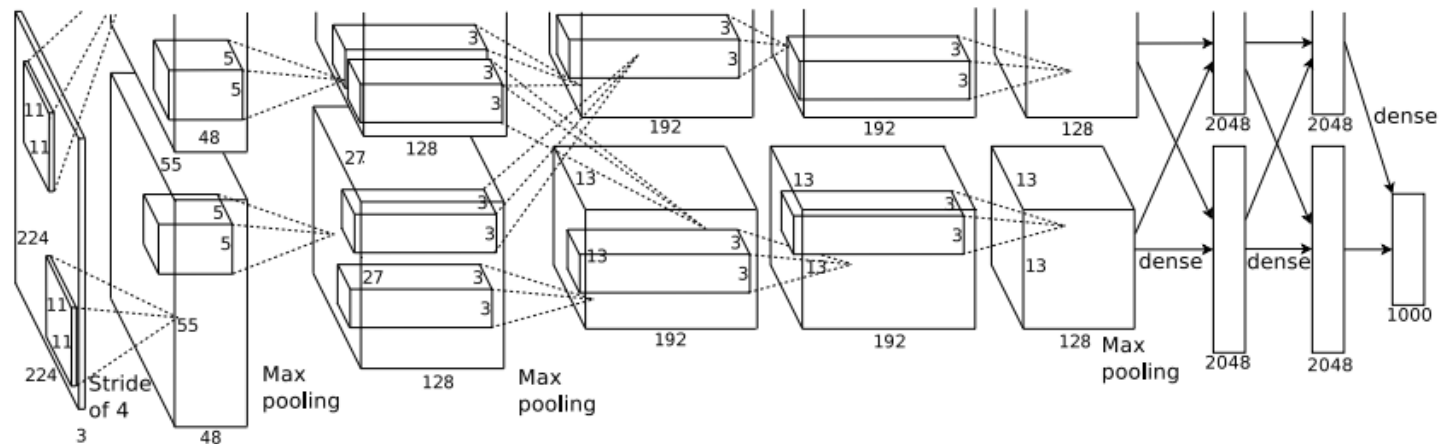
CONV5

Max POOL3

FC6

FC7

FC8

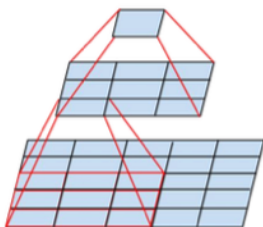


# Convolutional Neural Networks

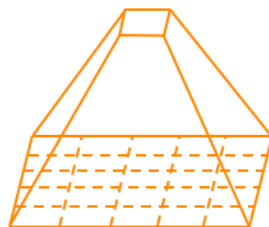
VGGNet : 더 깊은 신경망은 더 좋은 성능을 보인다! (Depth is matter!)

특징

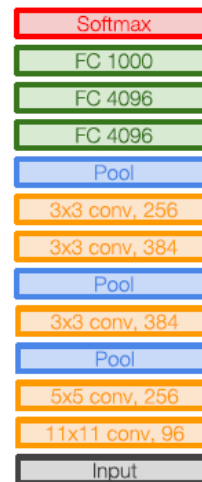
- 더 작은 필터 (3x3 conv)
- 더 깊은 네트워크 (11층, 13층, 16층, 19층)



two successive  
3x3 convolutions



5x5 convolution



AlexNet



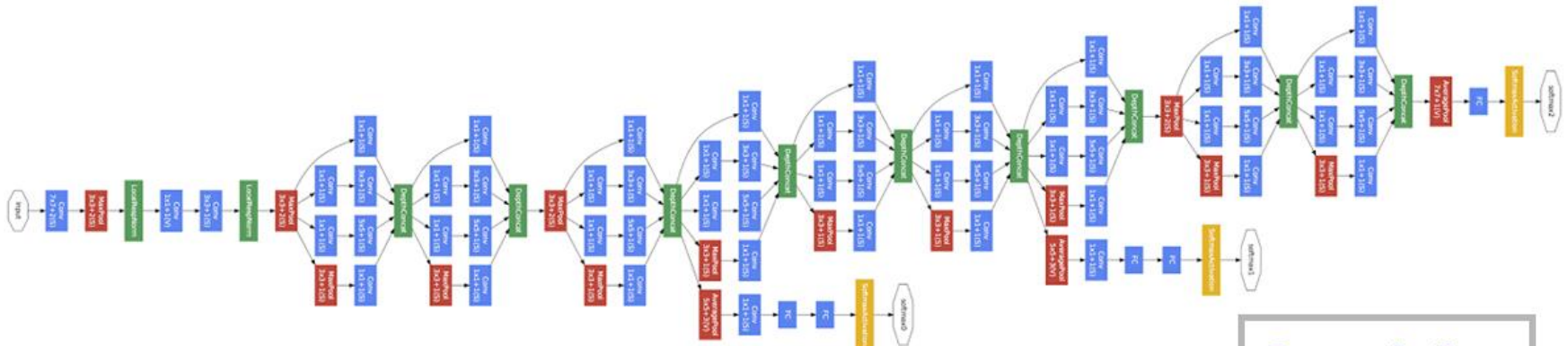
VGG16

VGG19

# Convolutional Neural Networks

## GoogleNet : 다양한 conv 연산의 활용

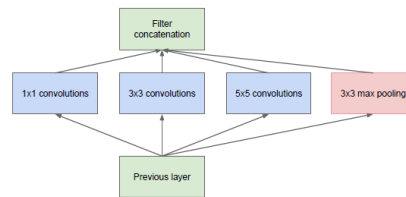
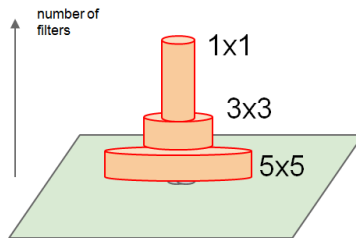
'14 ILSVRC 우승 Top-5 error : 6.7% (VGG: 7.3 %)



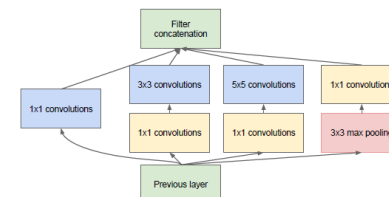
특징

- 더 깊고 더 복잡하다...! 22층!
- 다양한 사이즈의 conv 연산을 한번에 적용한 Inception module
- 한번에 좁은 영역, 넓은 영역을 동시에 보는 효과
- 1x1 conv로 차원축소 효과
- Auxiliary classifier 적용으로 vanishing gradient 문제 해결

Convolution  
Pooling  
Softmax  
Other



(a) Inception module, naïve version



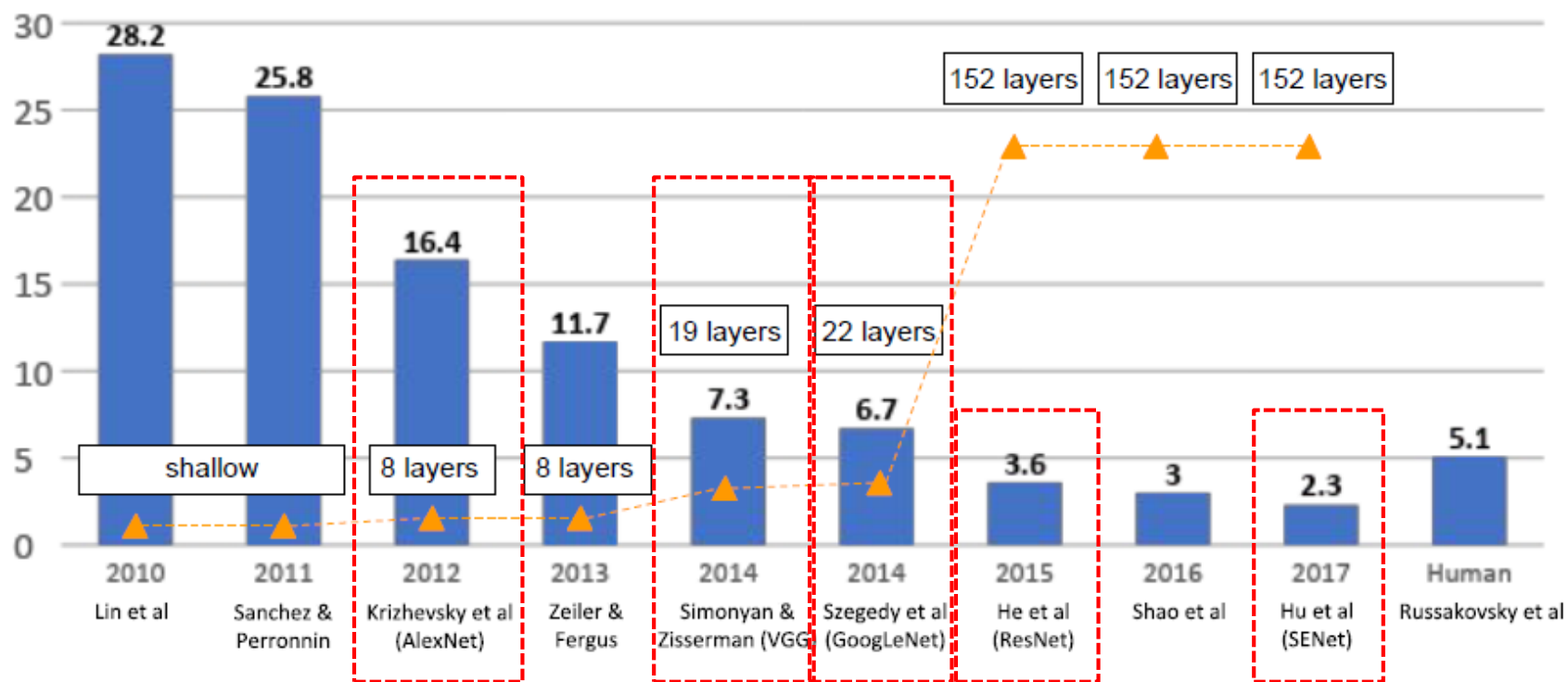
(b) Inception module with dimensionality reduction

Going deeper with convolutions, by Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2014).

# Convolutional Neural Networks

이제부터 인공지능이 사람의 성능을 뛰어넘기 시작합니다...

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



# Convolutional Neural Networks

ResNet : ???:"야 그렇게 몇층씩 늘려서 되겠냐? 팍팍 쌓자!"

'15 ILSVRC Top-5 accuracy : 3.57%



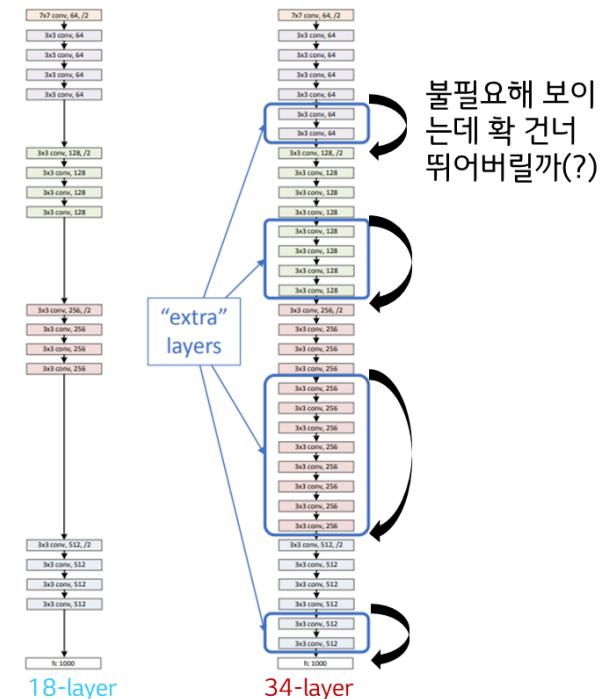
## 특징

- 최초로 사람의 성능을 뛰어넘은 인공지능 모델
- 무려 152층!!
- 무작정 깊다고 좋은것도 아니었다  
→ Residual connection 적용, 일종의 지름길 효과

## MSRA @ ILSVRC & COCO 2015 Competitions

### • 1st places in all five main tracks

- ImageNet Classification: *"Ultra-deep"* (quote Yann) **152-layer** nets
- ImageNet Detection: **16%** better than 2nd
- ImageNet Localization: **27%** better than 2nd
- COCO Detection: **11%** better than 2nd
- COCO Segmentation: **12%** better than 2nd



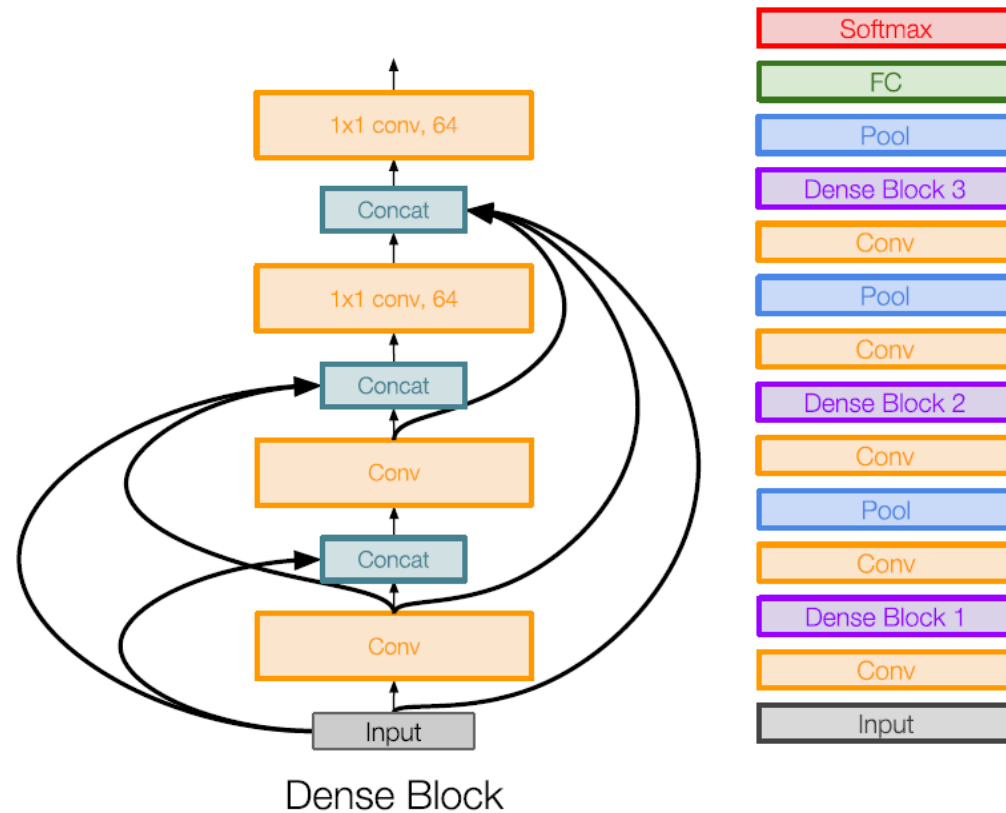


# Convolutional Neural Networks

DenseNet : 지름길이 좋다고?? 그럼 모든 부분에 넣자!

## 특징

- 가능한 모든 부분에 shortcut 적용
- 입력 데이터의 흐름 및 gradient의 흐름을 원활하게 하는 효과





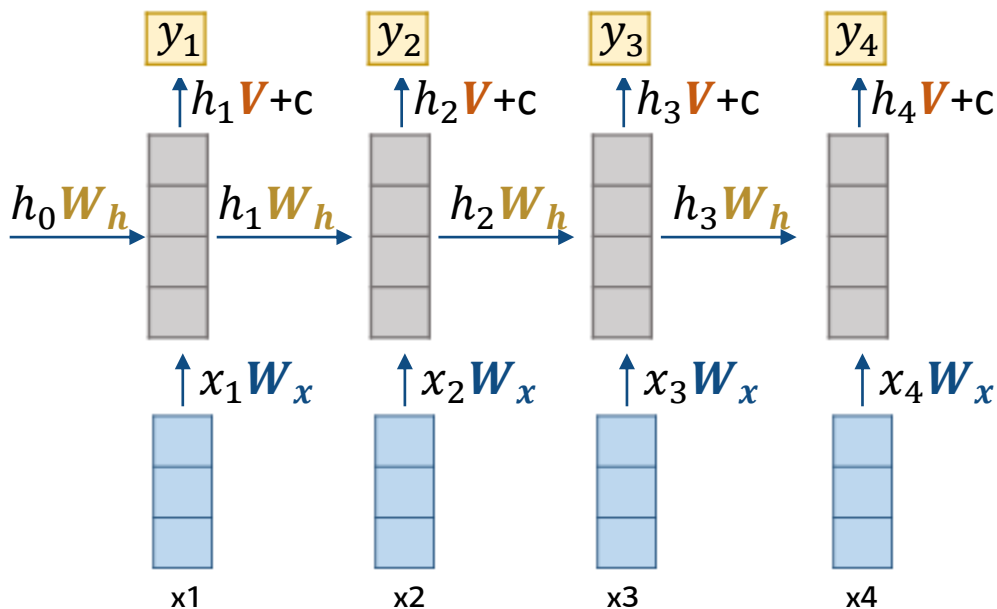
# Recurrent Neural Network

EXAMPLE : 어제 주가, 어제 KOSPI, 뉴스 언급량을 사용해 오늘의 주가를 예측해보자

Date	03-01	03-02	03-03	03-04
어제 주가	1000	1010	1200	1115
어제 KOSPI	2211	2220	2219	2200
뉴스 언급량	90	100	98	80
오늘 주가	1010	1200	1115	?

## RNN

- 매 시점 데이터를 처리할 때마다 동일한 파라미터( $W_x, W_h, V, b, c$ ) 공유
- 임의 길이의 sequential 입력 처리 가능



현 시점  $t$ 의 정보(hidden)는  
 현 시점  $t$ 의 input과  
 이전 시점  $t-1$ 의 정보(hidden)로 구성

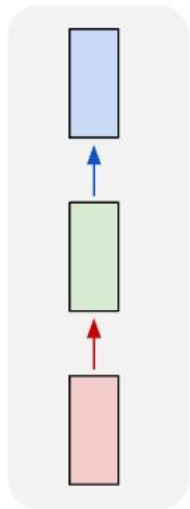
$$h_t = \tanh(x_t W_x + h_{t-1} W_h + b)$$

$$y_t = h_t V + c$$

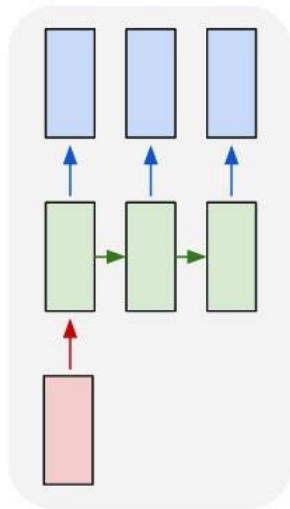
# Recurrent Neural Network

RNN은 입력과 출력의 길이가 유연하기 때문에 다양한 모델을 설계할 수 있다

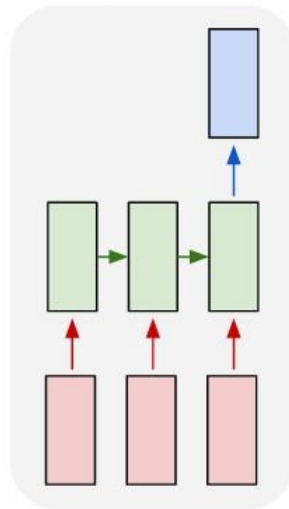
one to one



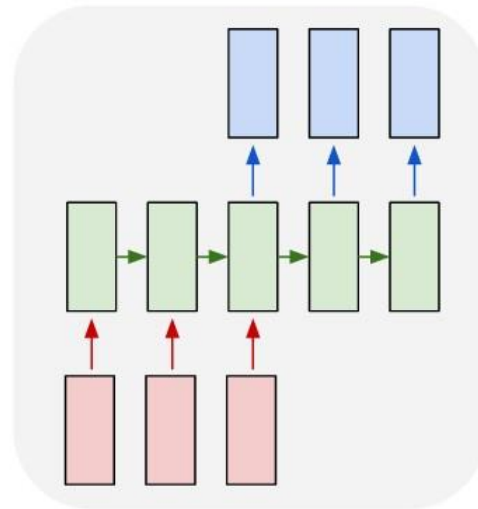
one to many



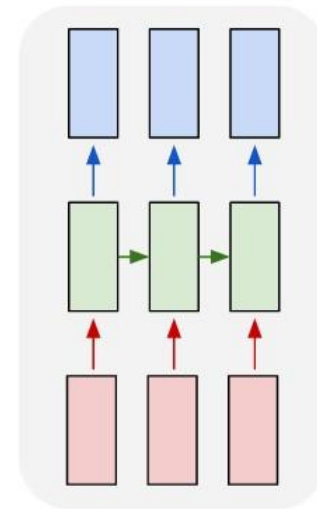
many to one



many to many



many to many



## RNN 구조의 장단점

### RNN 장점 (+)

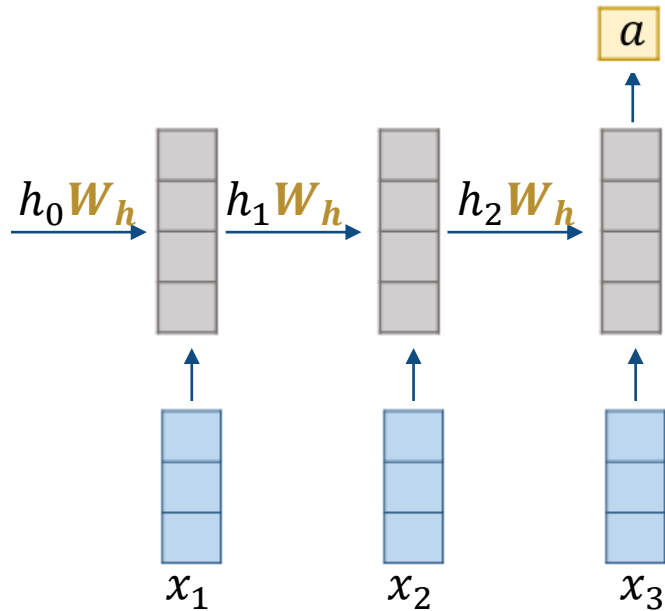
- 가변 길이의 입력 처리 가능
- 전 timestep에 걸쳐 파라미터 공유 → shared representations  
→ 긴 입력 값이 들어와도 모델 사이즈가 증가하지 않음
- (이론적으로) 많은 이전 단계의 정보를 현재의 timestep에 적용 가능

### RNN 단점 (-)

- 이전 timestep이 모두 계산되어야 현재 timestep 계산가능 → 다소 “느림”
- Gradient vanishing / exploding 현상
- (실질적으로) 많은 이전 timestep의 정보를 활용할 수 있는 모델이 아님  
→ long-term dependency

# Recurrent Neural Network

Backpropagation Through Time(BPTT) : RNN의 파라미터 업데이트하기



$$C = d(y, a)$$

$$a_t = h_t \mathbf{V} + c$$

$$h_t = \tanh(x_t \mathbf{W}_x + h_{t-1} \mathbf{W}_h + b)$$

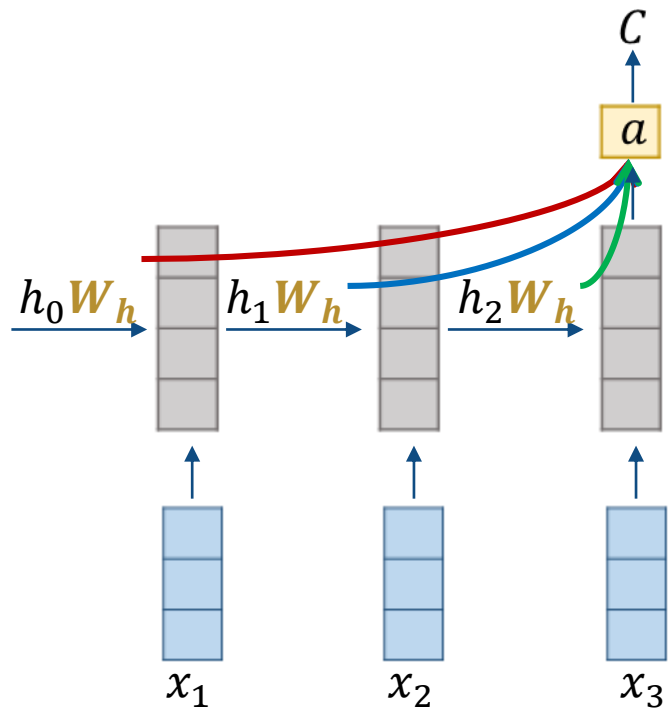
$W_h$ 를 업데이트 해보자!

REMIND :

$$w_{j+1} \leftarrow w_j - \alpha \frac{\partial C(w)}{\partial w_j}$$

# Recurrent Neural Network

Backpropagation Through Time(BPTT) : RNN의 파라미터 업데이트하기



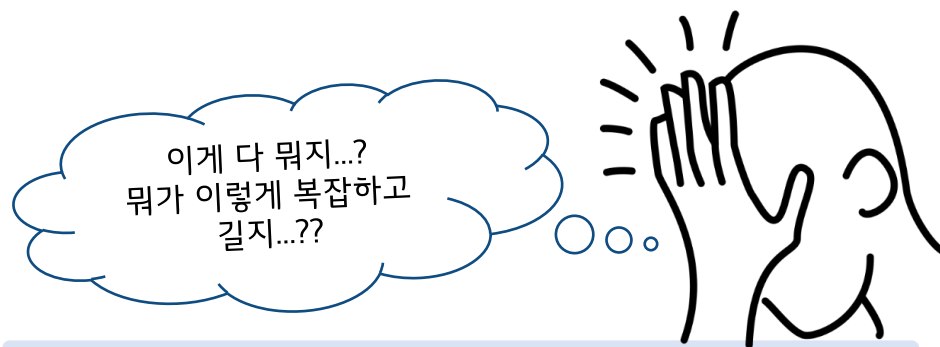
$$C = d(y, a)$$

$$a_t = h_t \mathbf{V} + c$$

$$h_t = \tanh(x_t \mathbf{W}_x + \underbrace{h_{t-1} \mathbf{W}_h}_{q_t} + b)$$

$\mathbf{W}_h$ 를 업데이트하기 위해, gradient를 구해보자!

$$\begin{aligned} \frac{\partial C}{\partial \mathbf{W}_h} = & \frac{\partial C}{\partial h_3} \frac{\partial h_3}{\partial q_3} \frac{\partial q_3}{\partial \mathbf{W}_h} + \\ & \frac{\partial C}{\partial h_3} \frac{\partial h_3}{\partial q_3} \frac{\partial q_3}{\partial h_2} \frac{\partial h_2}{\partial q_2} \frac{\partial q_2}{\partial \mathbf{W}_h} + \\ & \frac{\partial C}{\partial h_3} \frac{\partial h_3}{\partial q_3} \frac{\partial q_3}{\partial h_2} \frac{\partial h_2}{\partial q_2} \frac{\partial h_1}{\partial q_1} \frac{\partial q_1}{\partial \mathbf{W}_h} \end{aligned}$$



RNN에서는 timestep마다 weight를 공유,  
영향을 받은 모든 시간에서의 Loss를 더해줘야 한다

# Recurrent Neural Network

Gradient 정보가 점점 사라지거나 증폭되는 Gradient vanishing / exploding 현상

$$\frac{\partial C}{\partial W_h} = \frac{\partial C}{\partial h_3} \frac{\partial h_3}{\partial q_3} \frac{\partial q_3}{\partial W_h} +$$

$$\frac{\partial C}{\partial h_3} \frac{\partial h_3}{\partial q_3} \frac{\partial q_3}{\partial h_2} \frac{\partial h_2}{\partial q_2} \frac{\partial q_2}{\partial W_h} +$$

$$\frac{\partial C}{\partial h_3} \frac{\partial h_3}{\partial q_3} \frac{\partial q_3}{\partial h_2} \frac{\partial h_2}{\partial q_2} \frac{\partial h_2}{\partial h_1} \frac{\partial q_2}{\partial q_1} \frac{\partial h_1}{\partial q_1} \frac{\partial q_1}{\partial W_h}$$

여러 timestep을 고려하는 과정에서  
Gradient가 증폭될 수 있음  
(Gradient Exploding)

과거의 정보일수록 tanh미분값을 여러 번 곱하게 됨

→ 과거의 정보를 반영하는 부분의 gradient가 거의 0에 가까워짐 (Gradient Vanishing)

→ 현재와 먼 과거의 정보일수록 정보 의존성 감소

→ 즉, 현시점에서의 결정은 최근 내용위주로만 잘 반영, 오래된 시점의 내용은 거의 다 망각

# Recurrent Neural Network

Gradient vanishing / exploding 문제의 해결

## Gradient Vanishing

LSTM, GRU 등 변형된 Recurrent unit 활용

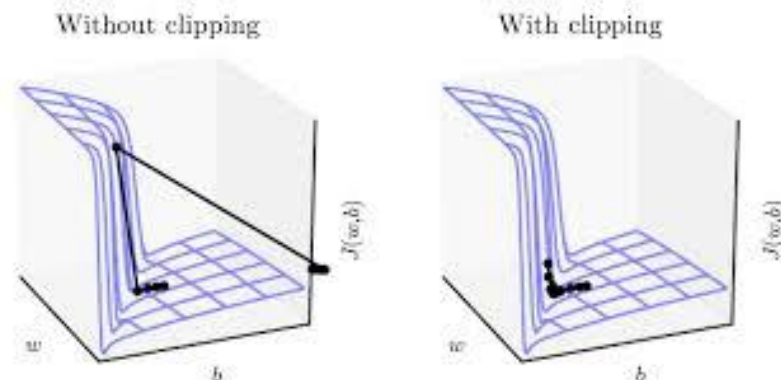
- 먼 과거의 정보도 잘 잊어버리지 않도록 변형
- 각 timestep의 입력들에 다른 가중치를 적용, 중요한 인풋의 가중치를 높임

Attention 메커니즘 활용

## Gradient Exploding

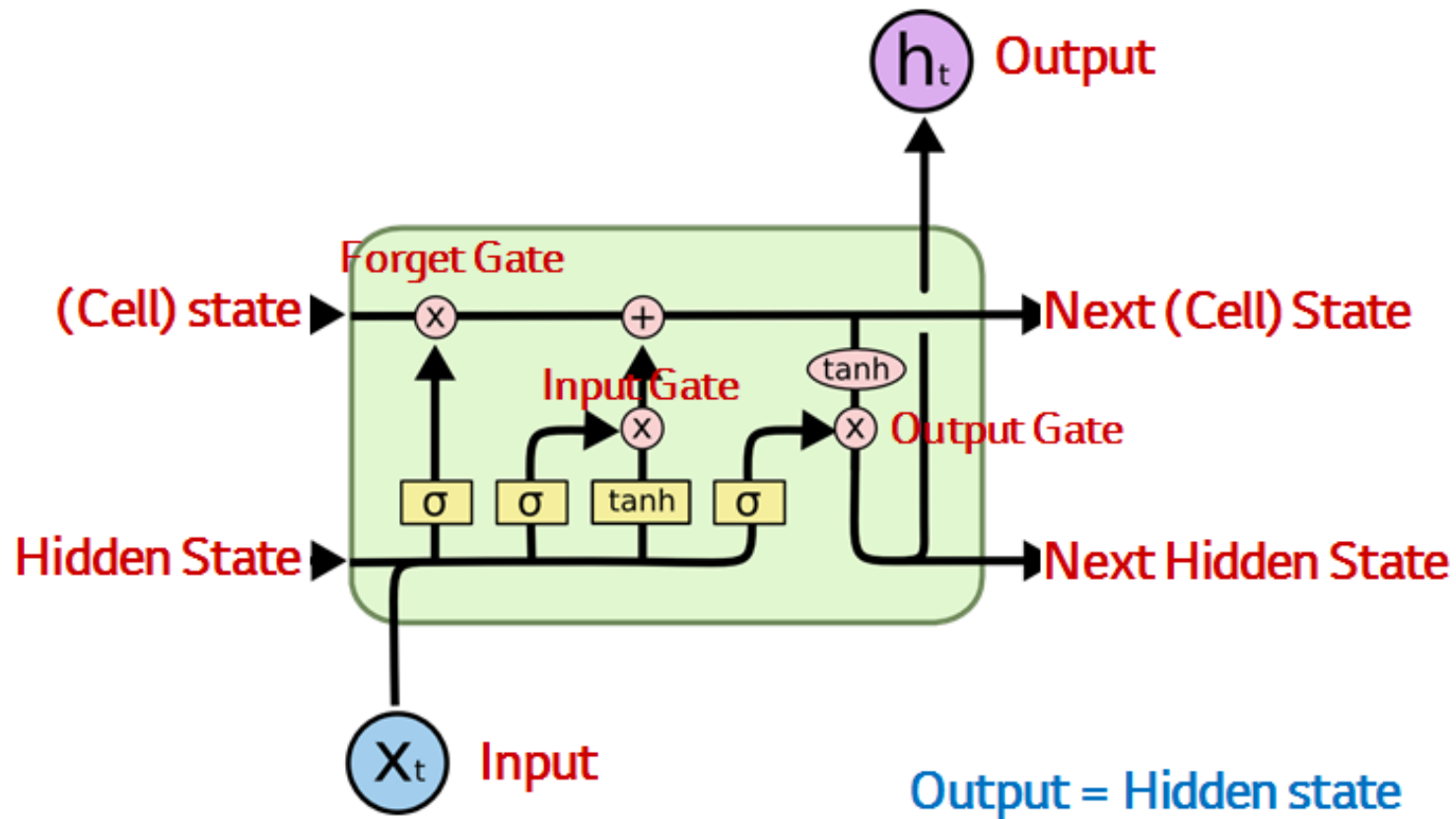
Gradient Clipping 활용

- Gradient가 너무 커질 경우, 지정된 상한선을 넘지 않도록 유지  
→ 파라미터를 너무 큰 폭으로 update하지 않도록 방지



# Recurrent Neural Network

LSTM (Long Short Term Memories) unit



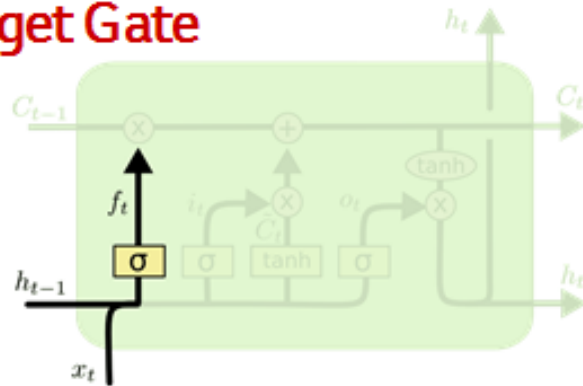
무섭고 복잡하게 생겼습니다만... 중요한 것은 Gate라는 것만 기억합시다!



# Recurrent Neural Network

## LSTM (Long Short Term Memories)

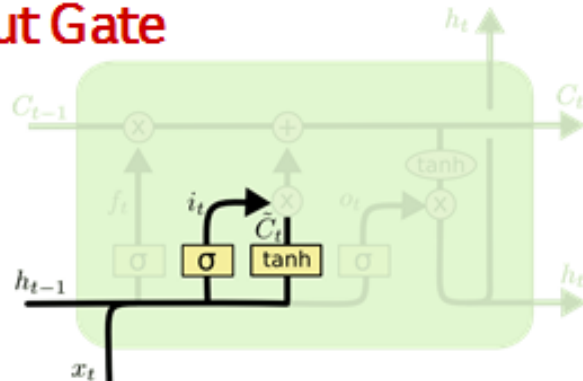
### Forget Gate



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

어떤 정보를 버릴지 결정

### Input Gate



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

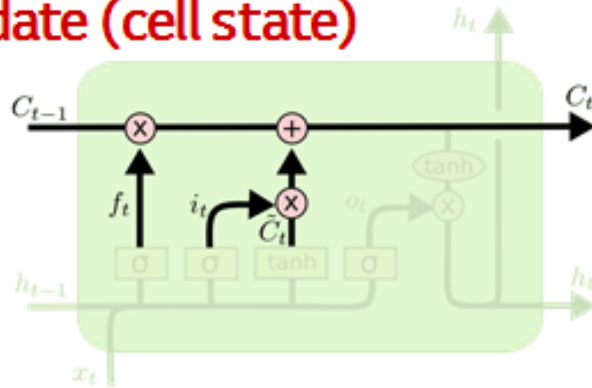
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

새로 들어온 정보 중 어떤 정보를  
얼마나 반영할 지 결정

# Recurrent Neural Network

## LSTM (Long Short Term Memories)

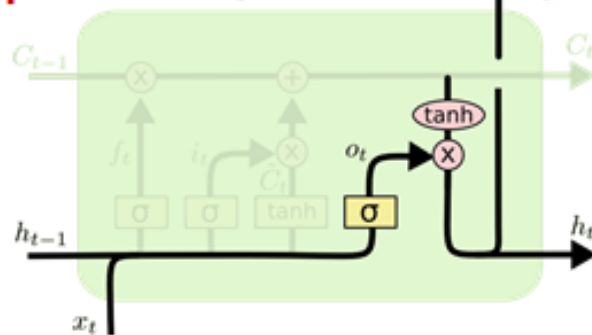
### Update (cell state)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

앞서 결정된 대로, 기존 정보와 새 정보의 반영 정도에 따라 업데이트

### Output Gate (hidden state)



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

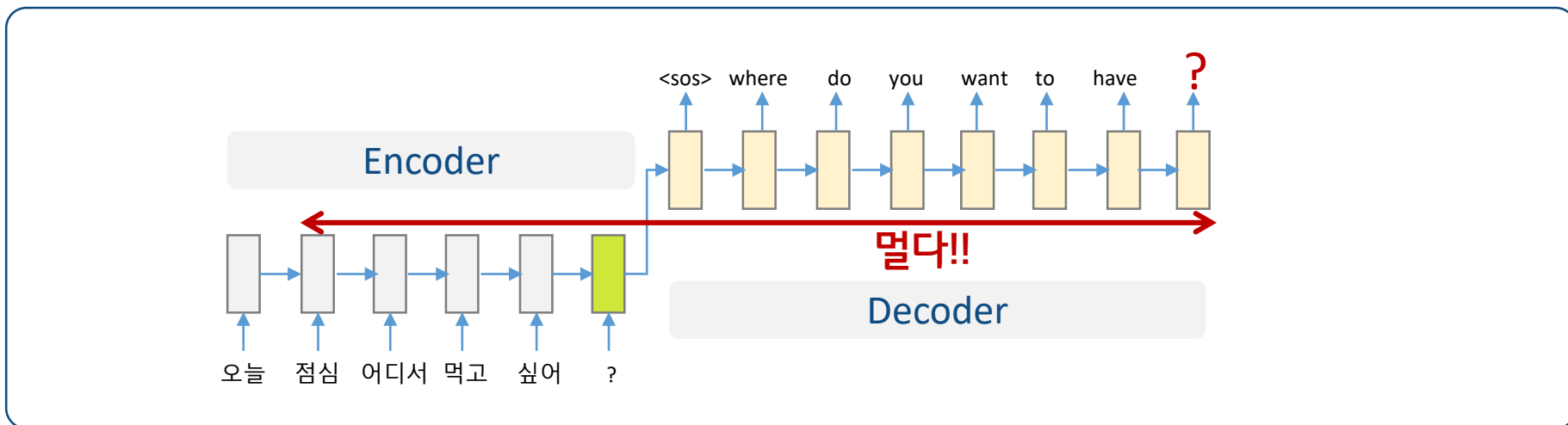
$$h_t = o_t * \tanh (C_t)$$

업데이트된 정보를 얼마나 반영하여  
output으로 내보낼지 결정

# Recurrent Neural Network

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기

- RNN을 번역 과제에 활용해보자 (feat. many-to-many)



- 인풋 문장을 읽은 RNN 인코더의 **마지막 hidden**은 문장의 전반적인 문맥 정보를 압축하여 담고 있음
- 그런데, 이 hidden은 먼 과거의 토큰 정보일수록 정보를 굉장히 압축해놓음 (**long term dependency** 문제)
- 따라서 위의 그림과 같이 마지막 hidden에만 의존해 전체 문장을 번역하려다 보면 원본에 어떤 내용이 있었는지 잊어버릴 수 있음

Motive

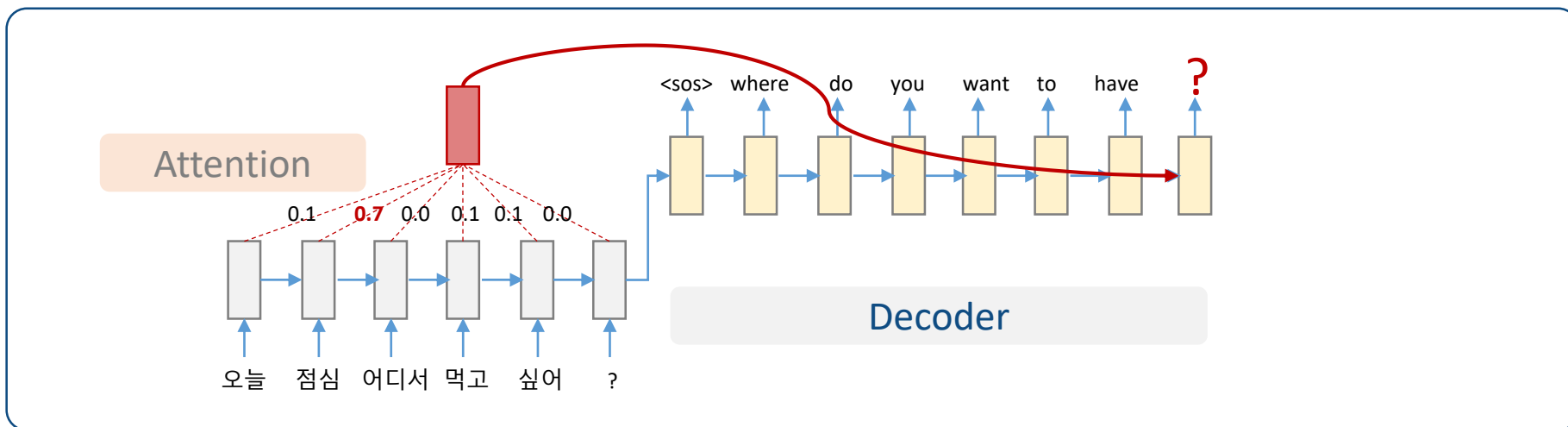
각 단어를 번역(decoding)할 때  
원본 문장에서 중요했던 단어와 문맥을 참고하면 나아지지 않을까?

# Recurrent Neural Network

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기

- Attention을 이용해 문맥 정보 알려주기

→ 각 단어를 번역할 때 원본에서 중요하게 봐야 할 단어와 문맥을 참고하게 만들자

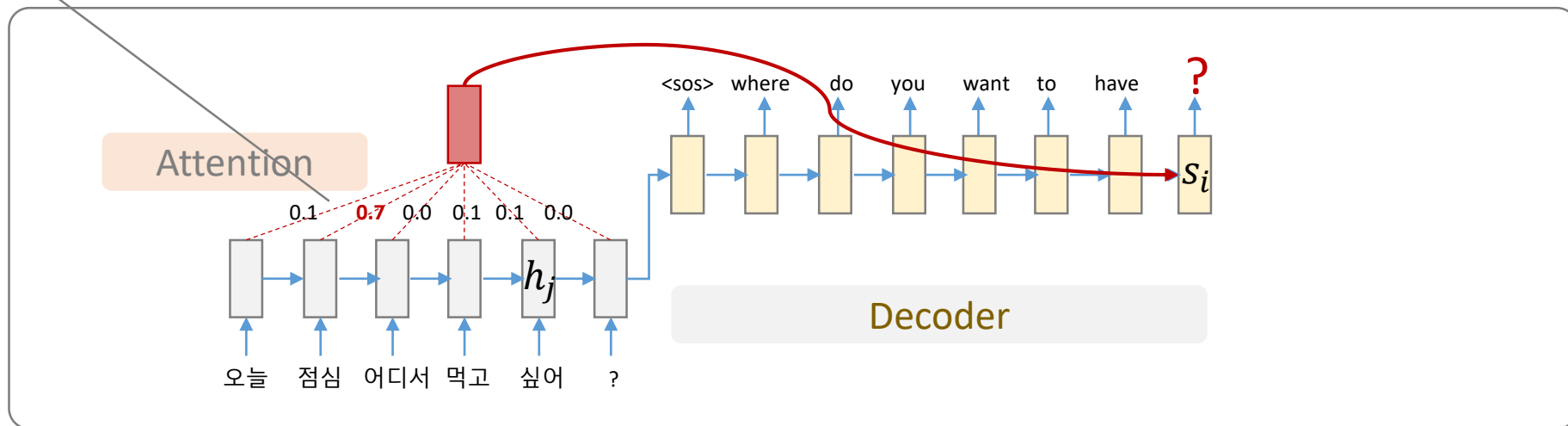


- 각 step에서 디코딩할 때 중요한 단어에 대해 집중된 feature를 생성해 RNN hidden에 추가해줌.
  - 예를 들어 위에서 ? 에 들어갈 단어를 번역 할 때는 원본 문장 중 <점심>에 집중하는 것이 좋으니 이에 해당하는 가중치 0.7로 가장 높게 계산된 벡터를 생성, 번역에 사용하면 <lunch>라는 올바른 단어를 꺼낼 수 있을 것
  - 가중치에 해당하는 attention score은 직전에 사용한 RNN 디코더 hidden과 input hidden과의 관련도 등으로 계산하기 때문에, 각 time step에서 다른 값을 가짐.
- 즉, 사람의 개입 없이 모델이 스스로 집중해서 봐야 할 포인트를 찾는다!

# Recurrent Neural Network

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기

Attention score을 계산하는 방법?



- 디코딩하는  $i$ 번째 타임스텝 직전의 hidden을  $s_{i-1}$ ,
- Attention 대상 토큰 중  $j$ 번째에 대한 hidden을  $h_j$  라고 할 때,  $i$ 번째 타임스텝의 hidden은 다음과 같이 구한다.

$$s_i = f(s_{i-1}, y_{i-1}, \overset{\text{context vector}}{c_i})$$

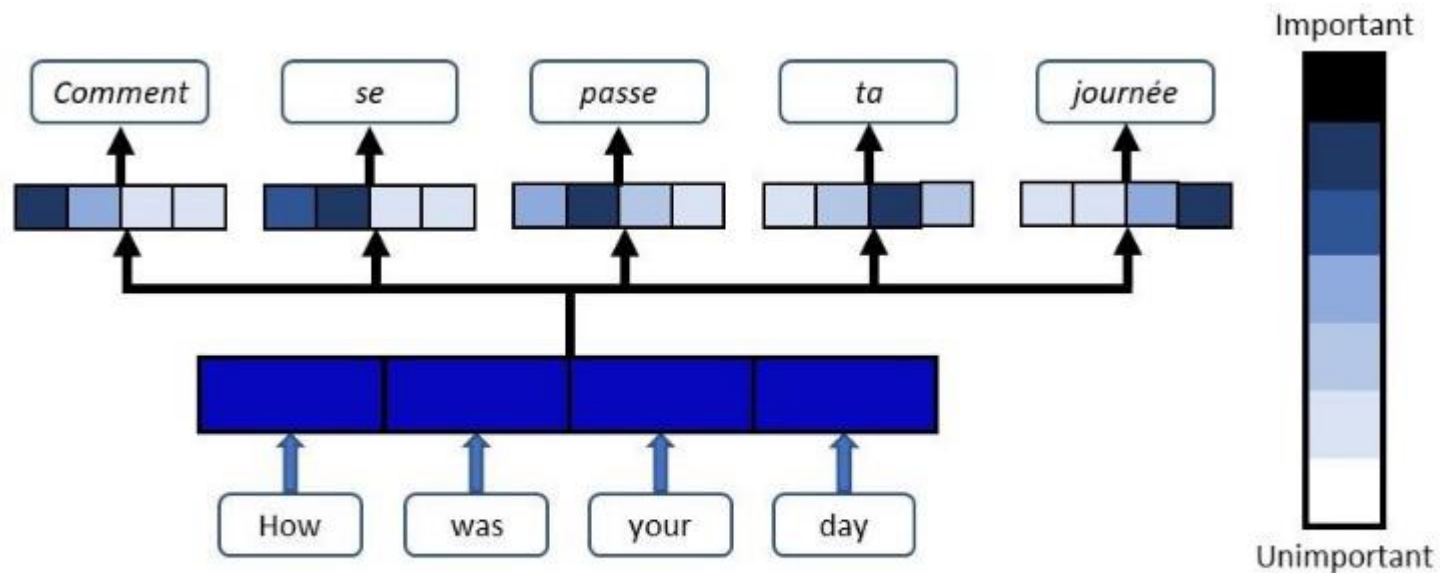
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

$$\text{where } c_i = \sum_{j=1}^{T_x} \overset{\text{attention score}}{\alpha_{ij}} h_j$$

$$e_{ij} = a(s_{i-1}, h_j) = \begin{cases} s_{i-1}^\top h_j \\ s_{i-1}^\top W_a h_j \\ v_a^\top \tanh(W_a [s_{i-1}^\top; h_j]) \end{cases}$$

# Recurrent Neural Network

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기



Weights are assigned to input words at each step of the translation

# Recurrent Neural Network

## Attention

- 번역 과제에서 고안된 개념이긴 하나, input의 hidden들 중 현재 timestep에서 중요한 부분에 가중치를 줘서 representation을 만들겠다는 attention의 개념은 이미지처리 등에서도 사용할 수 있음

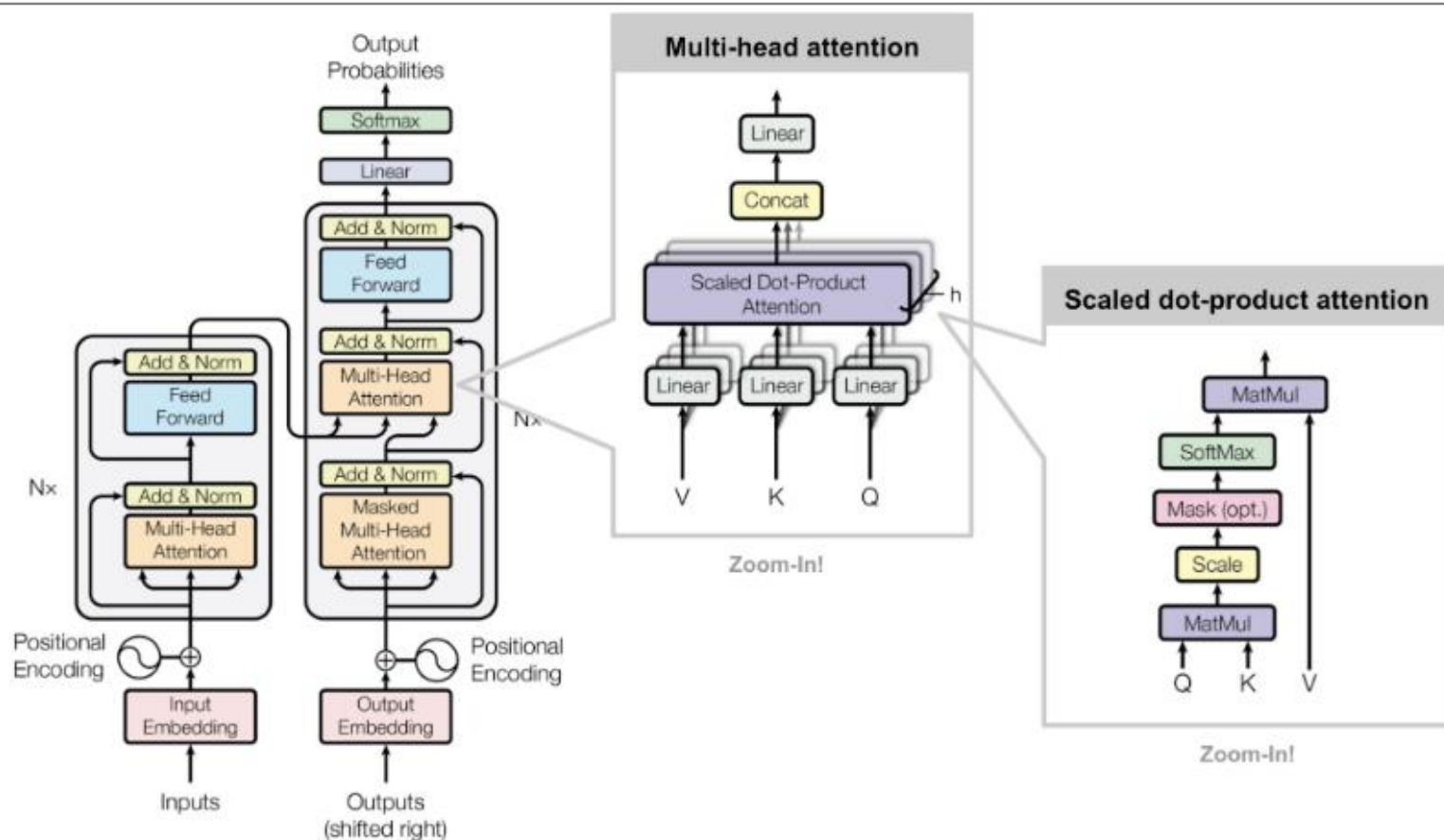
### <Show and Tell>



# Transformer

## 문장 시퀀스를 인코딩하는 새로운 접근법

- Transformer 구조를 제안한 “Attention is All you Need”는 2017년에 발표된 가장 흥미로운 논문 중 하나!
- Transformer에서는 **Self attention**을 사용해 Recurrent Unit 없이도 문장을 모델링할 수 있다.
- 핵심은 multi-head self-attention에서 사용하는 **scaled dot-product attention**



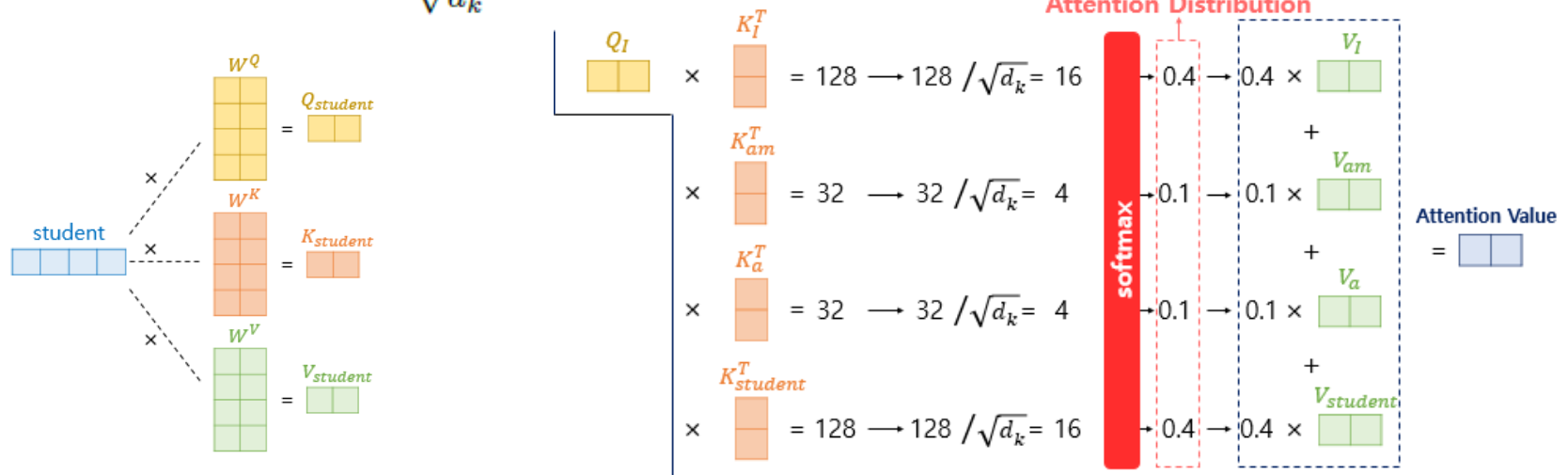


# Transformer

## Scaled dot-product attention

- Self attention은 인풋 시퀀스 전체에 대해 attention을 계산해 각 토큰의 representation을 만들어가는 과정으로, 업데이트된 representation은 문맥 정보를 가지고 있다.
- 예를 들어 “아이유는 1993년에 태어났다. 그녀는 최근에 드라마 호텔 델루나에 출연했다” 라는 인풋에 대해 self-attention을 적용하면 “그녀”에 해당하는 representation은 “아이유”에 대한 정보를 담게 된다.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Attention의 대상이 되는 토큰들을 key와 value, attention 하는 토큰을 query로 변환 (행렬곱)

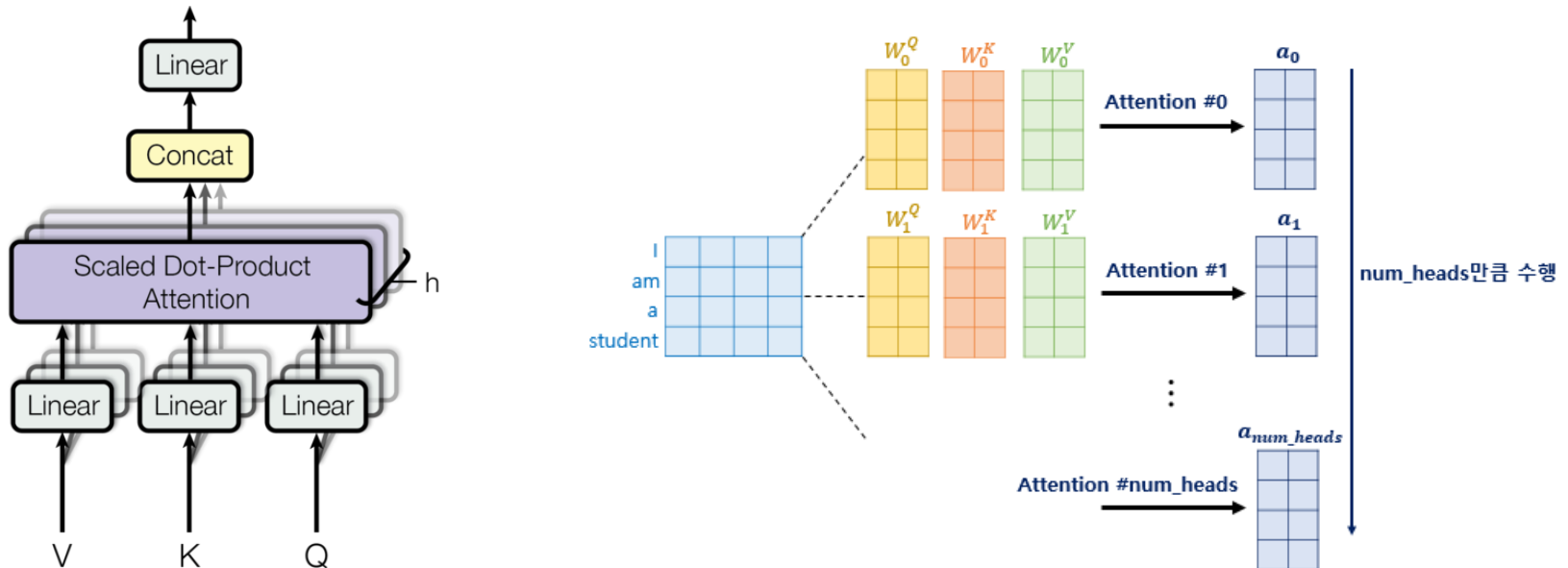
Attention 가중치는 query에 대해 각 key들의 가중치는 scale된 벡터 내적에 softmax를 취해서 구함.

가중치를 이용해 value를 가중합하여 query의 representation을 업데이트

# Transformer

## Multi head attention

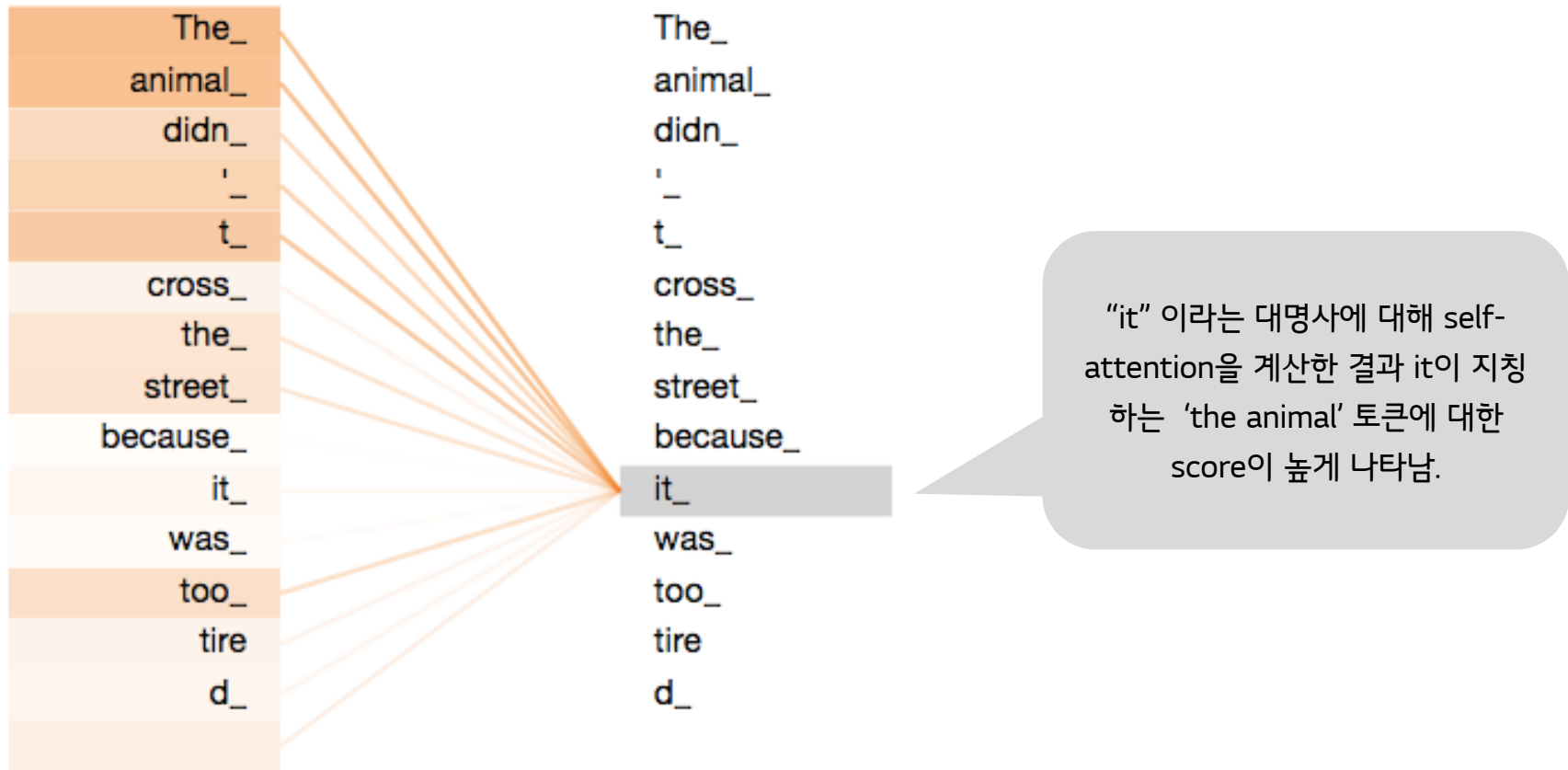
- Scaled dot-product attention을 한 번에 계산하는 것이 아니라 여러 개의 head를 이용해 계산함.
- 즉, 같은 attention 계산 과정을 여러 번 반복하여 그 결과를 concat하여 최종 attention score을 계산
- 이는 CNN filter을 여러 장 사용함으로써 이미지에 있는 다양한 특성을 포착하는 것처럼, 토큰 사이의 다양한 관계를 포착하기 위함임.



# Transformer

## Transformer Self-attention example

- “The animal didn’t cross the street because it was too tired”
- 라는 문장에 Transformer 구조를 이용해 self attention 적용



## Bidirectional Encoder Representations from Transformers (a.k.a BERT)

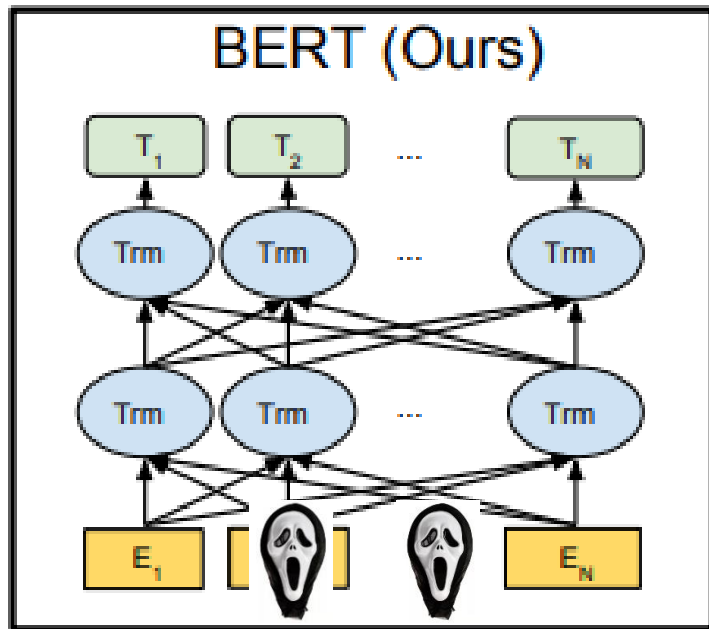
- Model 특징
  - Bi-directional
  - Transformer 구조를 여러 층 활용
  - 다량의 corpus로 사전학습
- 두 가지 사전학습 과제 수행시킨 뒤 fine-tuning
  - Masked Language Model
  - Next Sentence Prediction



# Transformer 활용 언어모델 : BERT

## 사전학습 과제 1: Masked Language Model

- 가려진 단어를 맞추는 과제를 해결함으로써 주변 맥락에 따른 단어의 의미 학습



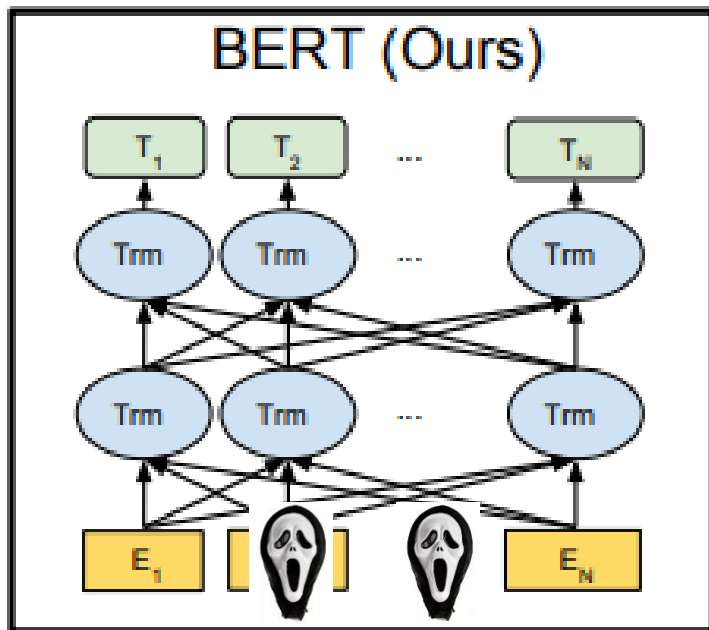
my dog is [MASK] →  → my dog is hairy

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	#ing	[SEP]
Token Embeddings	$E_{[CLS]}$	$E_{my}$	$E_{dog}$	$E_{is}$	$E_{cute}$	$E_{[SEP]}$	$E_{he}$	$E_{likes}$	$E_{play}$	$E_{ing}$	$E_{[SEP]}$
Segment Embeddings	+	$E_A$	$E_A$	$E_A$	$E_A$	+	$E_B$	$E_B$	$E_B$	$E_B$	+
Position Embeddings	$E_0$	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	$E_6$	$E_7$	$E_8$	$E_9$	$E_{10}$

# Transformer 활용 언어모델 : BERT

## 사전학습 과제 2 : Next Sentence Prediction

- 제시된 두 문장이 이어진 문장인지 아닌지를 맞추는 과제를 수행



문장 1 : 모자를 쓴 남성이 장바구니를 들고 마트에 갔다.  
문장 2 : 그 남자는 우유를 세 통 집어들었다



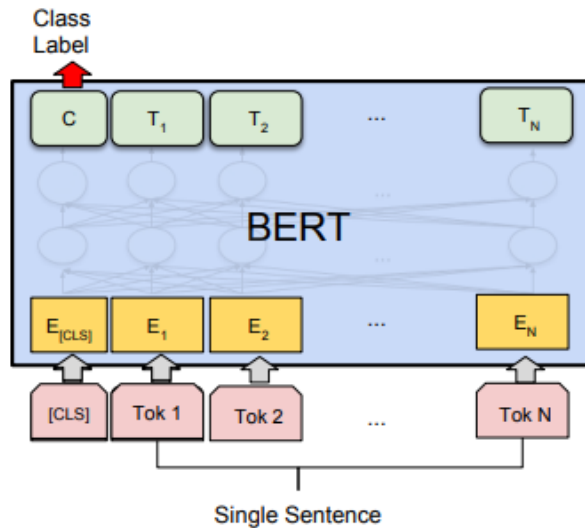
연결된 문장 맞음

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Token Embeddings	$E_{[CLS]}$	$E_{my}$	$E_{dog}$	$E_A$	$E_{cute}$	$E_{[SEP]}$	$E_{he}$	$E_{likes}$	$E_{play}$	$E_{##ing}$	$E_{[SEP]}$
Segment Embeddings	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_A$	$E_B$	$E_B$	$E_B$	$E_B$	$E_B$
Position Embeddings	$E_0$	$E_1$	$E_2$	$E_3$	$E_4$	$E_5$	$E_6$	$E_7$	$E_8$	$E_9$	$E_{10}$

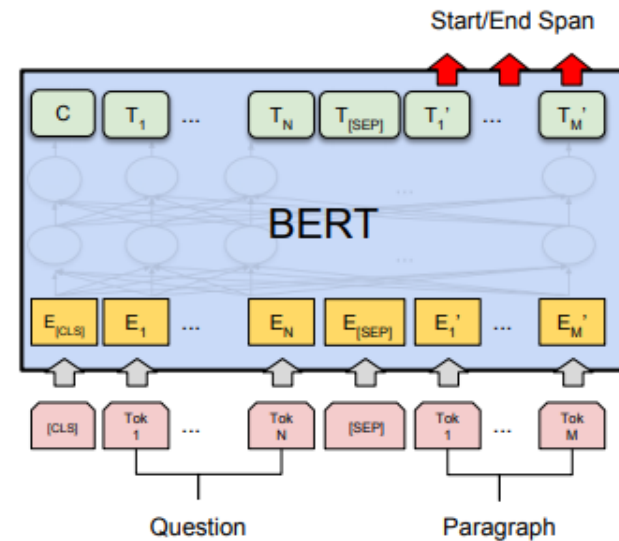
# Transformer 활용 언어모델 : BERT

## BERT Fine-tuning

- 주어진 과제 유형에 따라서 마지막 output layer만 변경하여 간단하게 fine-tuning



(b) Single Sentence Classification Tasks:  
SST-2, CoLA



(c) Question Answering Tasks:  
SQuAD v1.1

BERT : 방대한 양의 데이터로 수행한 사전학습 과제의 힘

- 사전 학습 과제

- 40 epoch, 1,000,000 iterations
- BERT<sub>base</sub>: 4 Cloud TPUs(=16 TPU chips)
- BERT<sub>large</sub>: 16 Cloud TPUs(= 64 TPU chips)
- 엄청난 자원을 사용하여 4일 내내 학습

- Fine-tuning

- 3~4 epoch만 추가 수행
- 추가학습은 조금만 수행해도 좋은 성능!!! → 사전학습의 위력



- 11개의 Natural Language Processing task에서 State-Of-The-Art 달성 !