[5차시 수업]

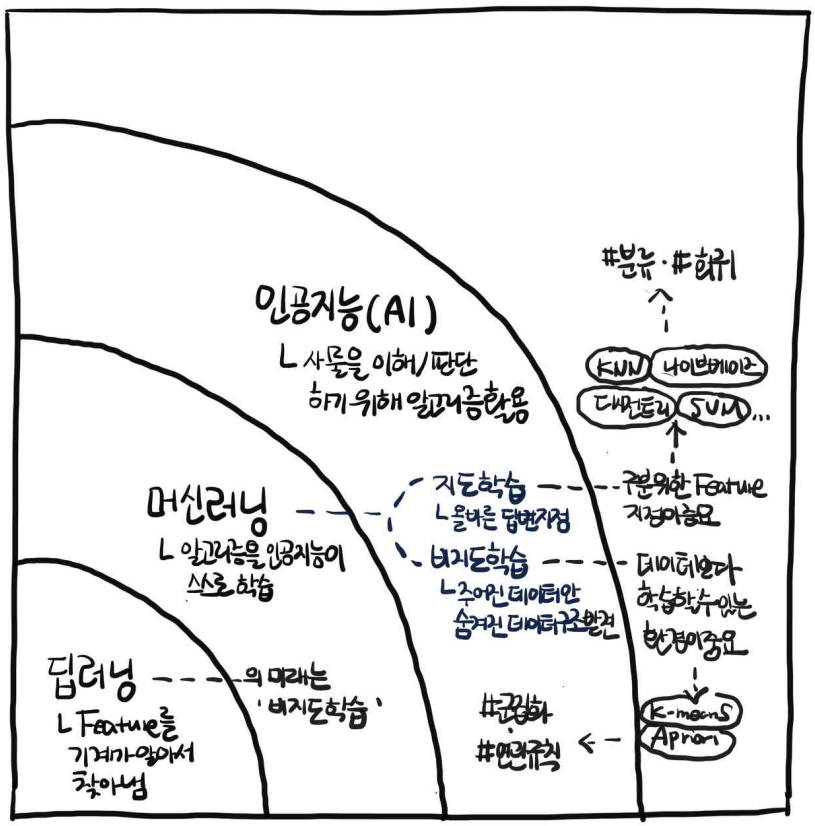
1) 딥러닝의 기본개념; 딥러닝의 이해, 딥러닝의 시작과 XOR 문제
- 실습: AND,_OR,_XOR_연산_신경망_만들기.ipynb
- 실습: XOR 계산
- Backpropagation(오차역전파)
- 미분,편미분,체인룰(chain rule)

2) CNN
- Convolution의 output 크기 계산하기
- CNN주요 모델

3) RNN
- RNN
- 실습; RNN에 대하여(08_RNN.ipynb)
- 실습; RNN을 이용한 시계열 데이터 예측
- 실습; 셰익스피어 텍스트 생성 (RNN)

[1교시]

1. 인공 지능과 머신 러닝, 딥러닝



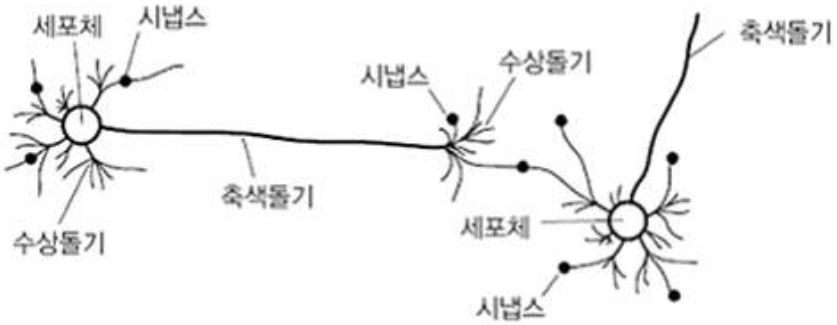
(인공지능의 구분)

1. 인공신경망

(人工神經網 / artificial neural network)

인공신경망이란, 인간의 뉴런 구조를 본떠 만든 기계학습 모델이다.

생물의 신경망, 특히 인간의 시각/청각 피질을 본떠 만든 알고리즘이다.

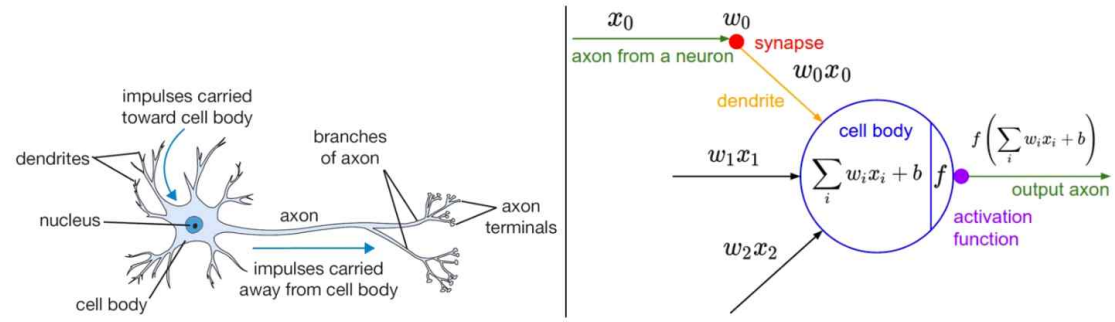


(뉴런(neurons)구조)

- 수상돌기; 다른 신경세포들이 보내는 신호를 전달받는 부분(Input)

- 축색돌기; 세포체로부터 아주 길게 뻗어가는 부분, 다른 신경세포에 신호를 전달하는 부분(Output)
- 시냅스; 신경세포들 사이의 신호를 전달해주는 축색돌기와 수상돌기 간을 연결.
신경세포의 신호를 무조건 전달하는 것이 아니라, 신호 강도가 일정한 값(임계치, Threshold) 이상이 되어야 신호를 전달하는 것으로, 각 시냅스마다 연결강도가 다를 뿐만 아니라 신호를 전달할지 말지를 결정하게 되는 것이다.

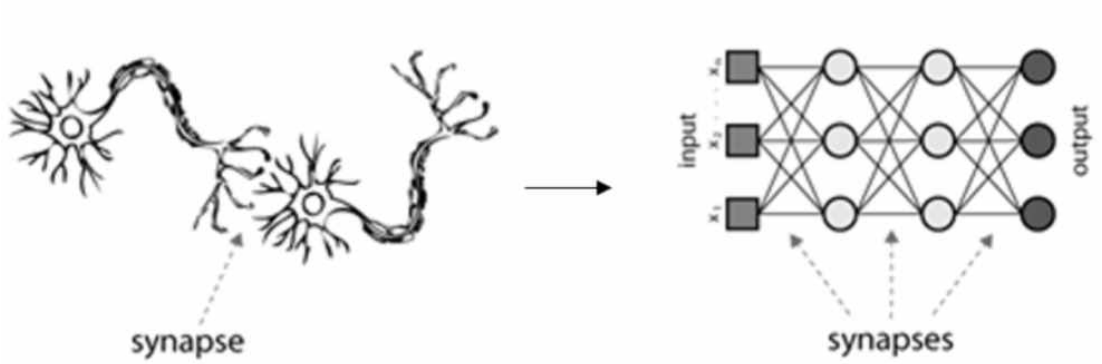
생물학적 신경망	인공신경망
세포체	노드(Node)
수상돌기(dendrite)	입력(Input)
축삭(Axon)	출력(Output)
시냅스	가중치(Weight)



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

(처리 단위(Processing Unit) : Neuron vs. node)

• x0, x1, x2 : 입력되는 뉴런의 축삭돌기로부터 전달되는 신호의 양
• w0, w1, w2 : 시냅스의 강도, 즉 입력되는 뉴런의 영향력을 나타냅니다.
• w0x0 + w1x1 + w2*x2 : 입력되는 신호의 양과 해당 신호의 시냅스 강도가 곱해진 값의 합계
• f : 최종 합계가 다른 뉴런에게 전달되는 신호의 양을 결정짓는 규칙, 활성화 함수



(연결(connection) : Synapse vs. weight)

- 인간의 신경세포가 하나가 아닌 다수가 연결되어 의미 있는 작업을 하듯,
- 인공신경망의 경우도 개별 뉴런들을 시냅스를 통해 서로 연결시켜서 복수개의 계층(layer)을 연결하고, 각 층간의 연결 강도는 가중치로 수정(update) 가능하다.

2. 인공신경망 기본원리

- 몇 개의 층위를 만들어서 그 안에 '세포'들을 집어넣고, 이들을 무작위 연결 강도로 연결
- 각 '세포'들은 자신에게 들어온 신호를 가중치와 곱해 모두 더하고(wxwx), 역치와 비교해서(wx + bwx+b) 신호를 다음 뉴런으로 전달한다.
- 전송되는 신호는 입력 신호의 **선형 합**이고, 선형 합은 입력 신호의 선형 합.

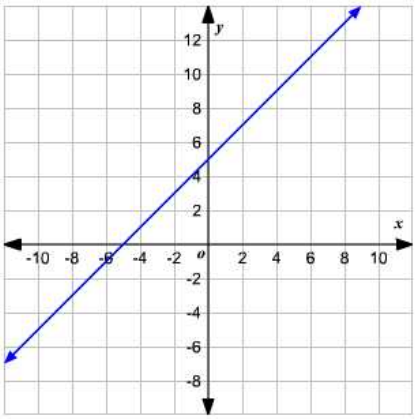
다시 말해 **아무리 많은 층을 쌓아봤자 행렬 곱하기 연산을 한 번 한 것과 동일**한 결과가 나오는 것

- 이와 같은 문제점은 **비선형 활성 함수를 도입**하여 해결할 수 있다. 비선형 함수를 도입할 경우 인공신경망 모델로 비선형 문제를 풀 수 있고, 많은 층을 쌓을 경우 대체적으로 결과물이 향상된다.

1) 선형함수

선형함수란 그래프를 그렸을 때 직선의 형태를 갖는 것.

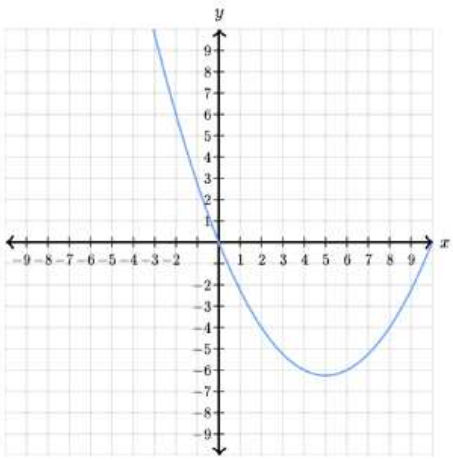
선형함수는 함수 값 예측이 쉽다.



(2) 비선형함수

비선형함수는 그래프를 그렸을 때 직선의 형태를 갖지 않는 것.

비선형 함수는 함수 값을 예측하기가 매우 어렵다.



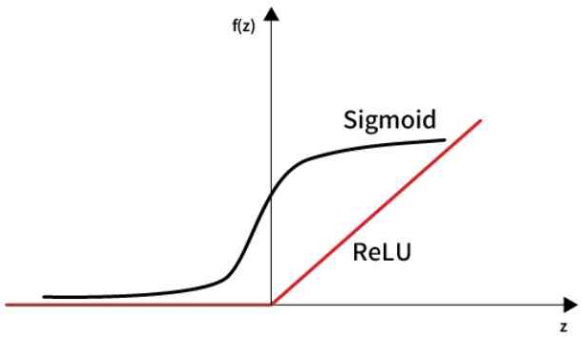
3. 딥러닝 개요

1) 딥러닝의 겨울

- 1957년 프랑크 로젠블라트(Frank Rosenblatt)가 퍼셉트론 이론을 발표한 이래 인공신경망 이론은 효과적인 학습 모델을 찾지 못한 채 여러 가지 문제에 직면 하였다.
- 역전파(backpropagation)에서 인공신경망의 레이어(layer)가 늘어날수록 데이터에서 기울기가 사라지는 문제(vanishing gradient problem)와 학습 데이터를 과하게 학습하여 학습 데이터에 대해서는 오차가 감소하지만 실제 데이터에 대해서는 오히려 오차가 증가하는 과적합(overfitting) 문제, 문제의 규모가 커질 때마다 나타나는 높은 시간 복잡도와 컴퓨터 성능의 한계 등으로 인해 인공신경망 이론은 큰 진전을 보지 못하고 정체 상태를 맞이하게 된 것을 딥러닝의 겨울이라고 한다.

2) 딥러닝의 부활

- 2006년 토론토 대학의 **제프리 힌튼(Geoffrey Hinton)** 교수는 심층 신뢰 신경망(Deep Belief Network, DBN)이라는 딥러닝에 매우 효과적인 알고리즘에 관한 논문을 발표.
- 제프리 힌튼 교수는 이 논문을 실제 적용하여 2012년 세계 최대 이미지 인식 경연대회인 ILSVRC에서 나머지 팀들이 26% 대의 이미지 인식 오류율을 보일 때, 홀로 15% 대의 오류율을 기록함으로써 1위를 차지
- 인공지능 분야의 전문가들 대부분은 제프리 힌튼 교수의 이 논문이 딥러닝의 부활을 알리는 계기가 되었다고 말하고 있다.
- 또한, 기울기가 사라지는 문제(vanishing gradient problem)를 해결하기 위해 기존에 사용하던 시그모이드(sigmoid) 함수 대신에 **ReLU(Rectified Linear Unit)**라는 함수가 새롭게 고안되었으며, **드롭아웃 계층(dropout layer)**을 사용하여 학습 중일 때 랜덤하게 뉴런을 비활성화 함으로써 학습이 학습 데이터에 치우치는 과적합(overfitting) 문제를 해결하였다.
- 이러한 알고리즘의 개발과 컴퓨터 하드웨어의 급속한 발달, GPU를 활용한 병렬처리 기술의 개발 등으로 딥러닝은 획기적으로 그 성능이 향상되어 새로운 도약의 시대를 맞이하게 되었음



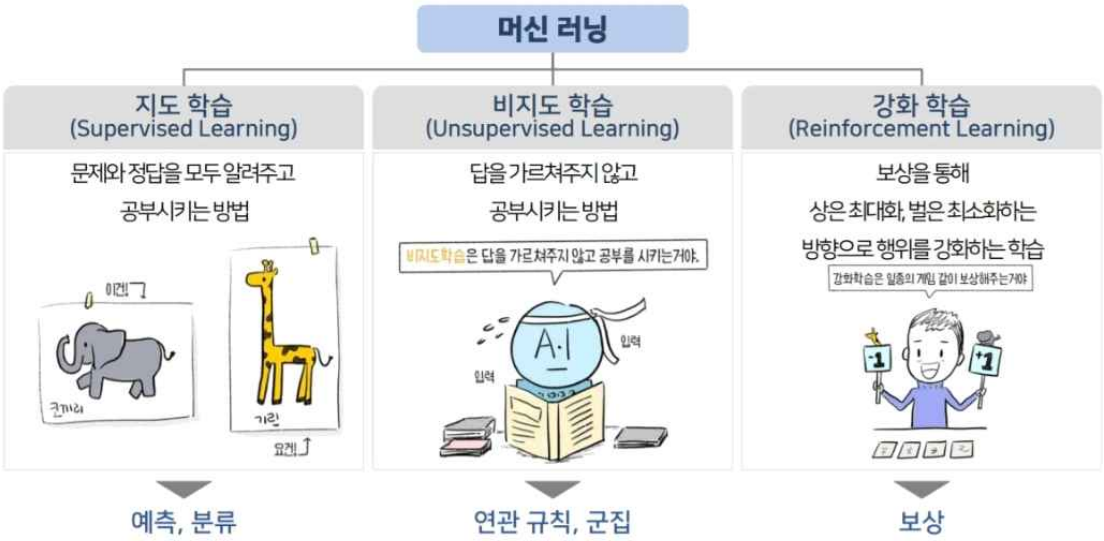
(시그모이드 함수와 ReLU함수)

3) 딥러닝의 도약과 그 원동력

1. GUP 기반의 병렬처리를 포함한 컴퓨팅 파워(computing power)의 발달
2. 인터넷을 통해 축적된 엄청난 양의 빅데이터(big data)
3. 딥러닝을 위한 획기적인 알고리즘의 고안

4. 딥러닝 이란

1) 머신러닝(기계학습)



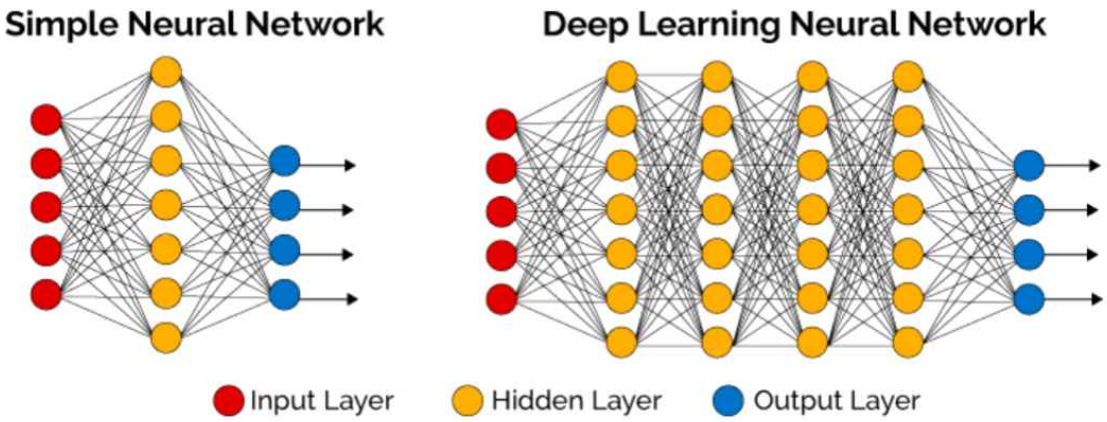
◈ 지도학습(Supervised Learning): 분류(Categorize)와 회귀(Regression)
예) 인구예측, 수수료 예측, MNIST를 통한 숫자 예측, 라벨링된 이미지 분류 등

◈ 비지도학습(Unsupervised Learning): 군집화(Clustering), 연관규칙(Association Rule)
예) 이상탐지 시스템(FDS), 동물 구분하기, 학습을 통한 추천(넷플릭스 추천 등)

◈ 강화학습(Reinforcement Learning): 보상을 통한 레벨업 (Reward)
예) 목표를 달성하면 보상, 실패하면 재시도 - 아타리의 벽돌게임

2) 딥러닝

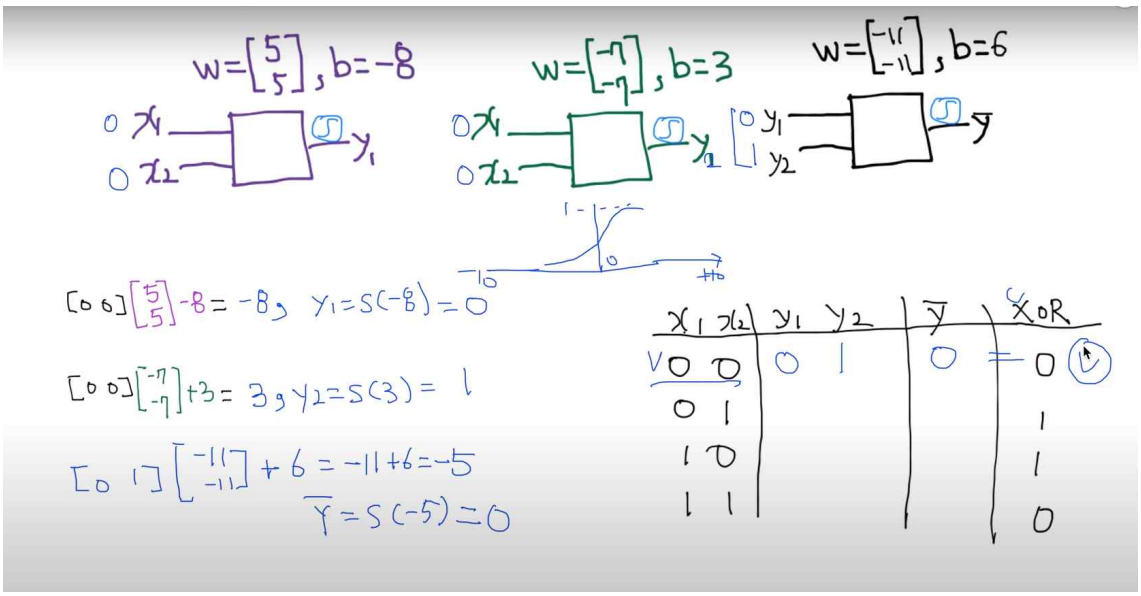
- 인간의 뇌구조를 모방한 뉴럴 네트워크
- 기존에는 하나의 레이어를 활용, 여러 번의 학습을 통해 최적화된 답을 내었다면, 여러개의 신경망, 히든 레이어들을 활용하여 심층신경망을 활용하는 방식
- 딥러닝은 예측하고자 하는 수치를 얻기 위해 Feature들을 히든 레이어(hidden layer)를 활용함으로써 보다 명확한 분류를 할 수 있고, 그 정확도가 기존에 비해 높다.

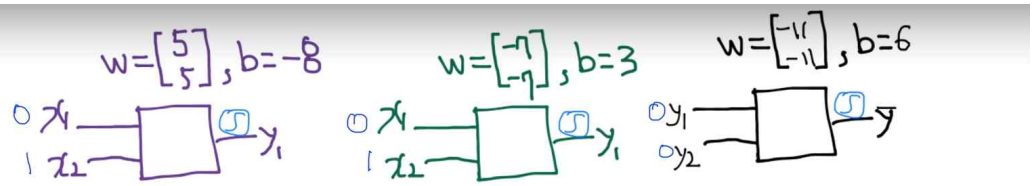


- 딥러닝은 학습하는 알고리즘 방식(히든 레이어를 사용하였는지 아닌지)의 지도, 비지도, 강화학습 모두 알고리즘에 히든레이어를 사용한다면 딥러닝이라고 부를 수 있다.
- 히든레이어(hidden layer)란, 인풋 레이어(input layer)와 아웃풋 레이어(output layer) 사이에 위치한 레이어로 입력 값을 넘겨받아 설정해둔 가중치(weight)에 따라 계산하여 아웃풋 값을 내게 되는 중간에 숨겨져 있는 것
- 히든레이어는 몇 개의 멀티 레이어를 사용하는지에 따라 정확도(Accuracy)가 달라지므로, 인공지능 알고리즘이 블랙박스 모델이라고 이야기하는 이유이기도 하다.
- 머신러닝 알고리즘은 히든 레이어를 몇 개를 사용할지, 또 가중치를 어떻게 줄 것인지(Regulation), 다양한 파라미터 값을 어떻게 조정할지에 따라 정확도(accuracy)가 달라진다.
- 따라서 미세한 수치들의 조정(Hyper Parameter Tuning)은 정확하게 알고리즘 모델을 이해하고 조정할 필요가 있다.

5. XOR 문제

1) XOR 계산





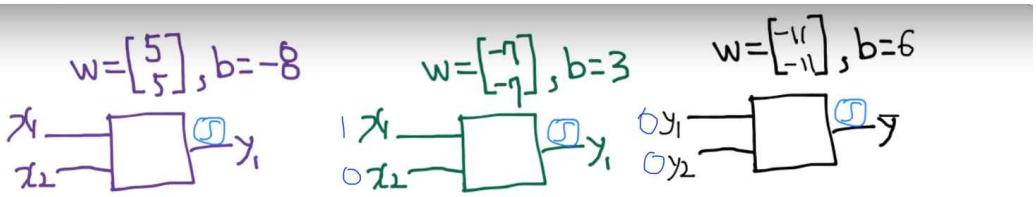
$[0\ 1]\begin{bmatrix}5\\5\end{bmatrix}-8=0+5-8=-3$, sigmoid$(-3)=0$

$[0\ 1]\begin{bmatrix}-7\\-7\end{bmatrix}+3=0-7+3=-4$, sigmoid$(-4)=0$

$[0\ 0]\begin{bmatrix}-11\\-11\end{bmatrix}+6=0+0+6=6$

sigmoid$(6)=1$

x_1	x_2	y_1	y_2	y	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0				1
1	1				0



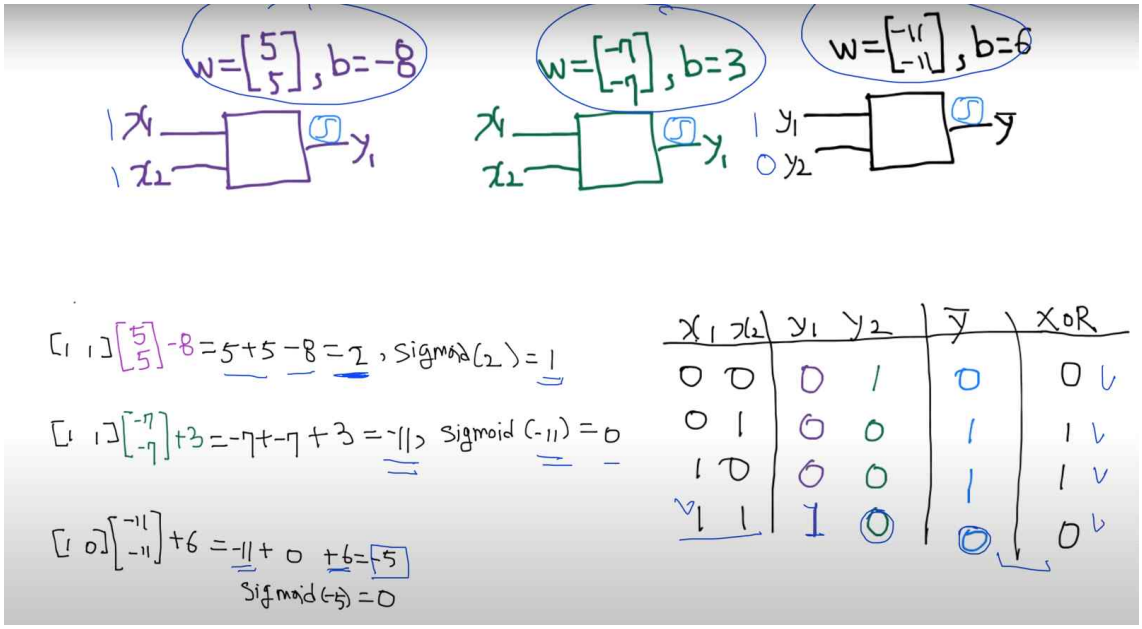
$[1\ 0]\begin{bmatrix}5\\5\end{bmatrix}-8=5+0-8=-3$, sigmoid$(-3)=0$

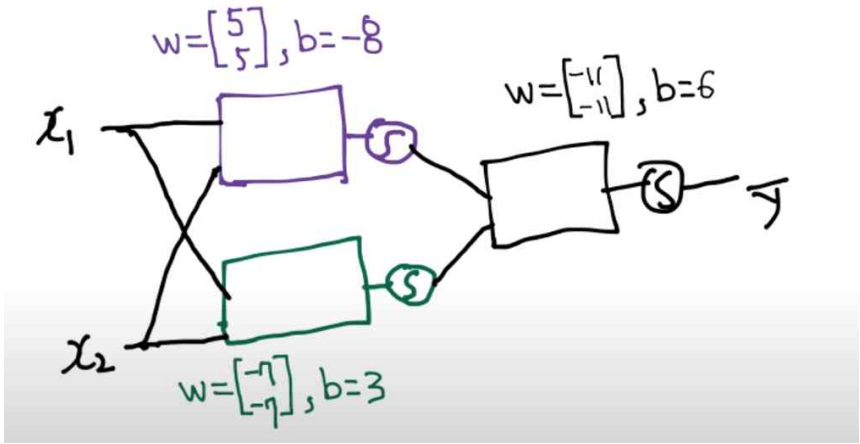
$[1\ 0]\begin{bmatrix}-7\\-7\end{bmatrix}+3=-7+0+3=-4$, sigmoid$(-4)=0$

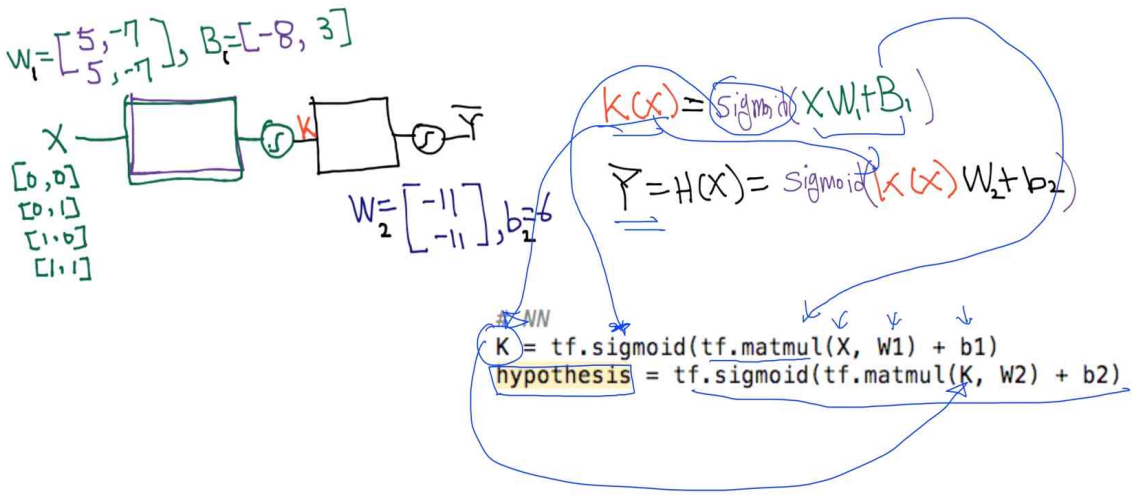
$[0\ 0]\begin{bmatrix}-11\\-11\end{bmatrix}+6=0+0+6=6$

sigmoid$(6)=1$

x_1	x_2	y_1	y_2	y	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1				0 ?

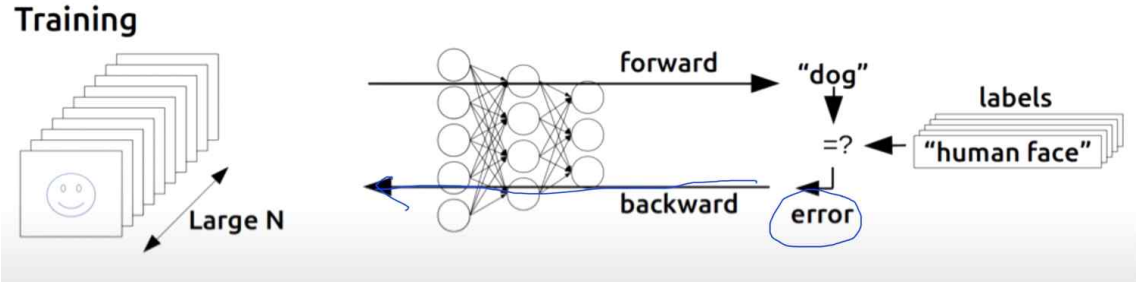


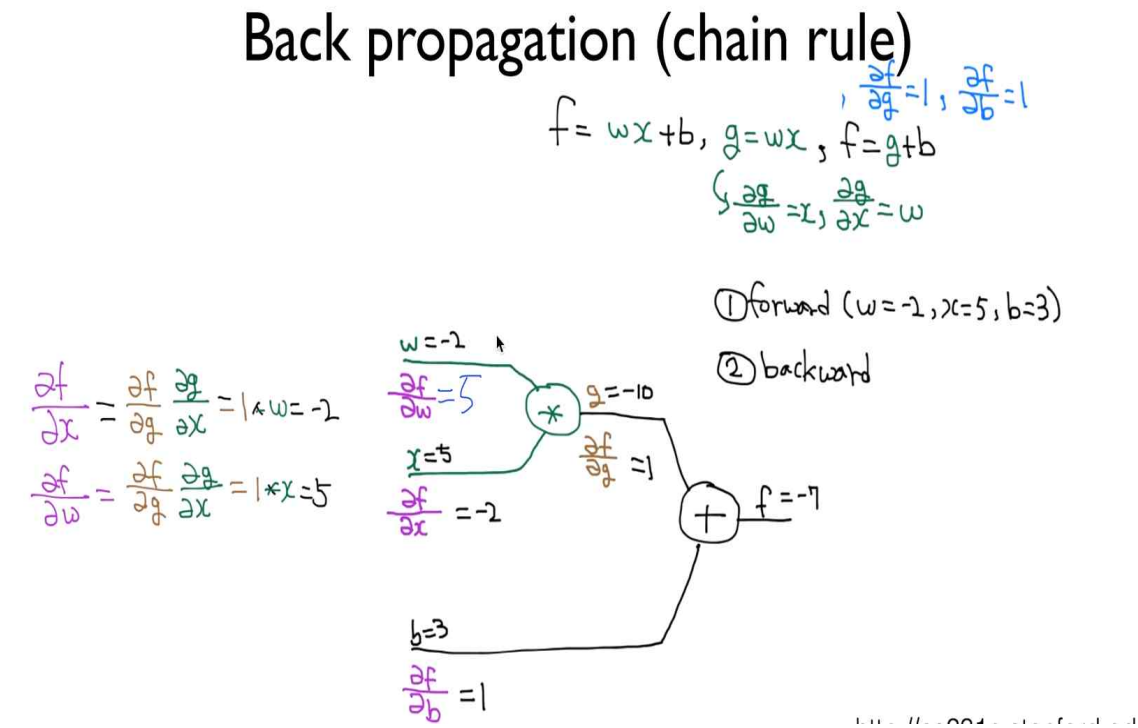


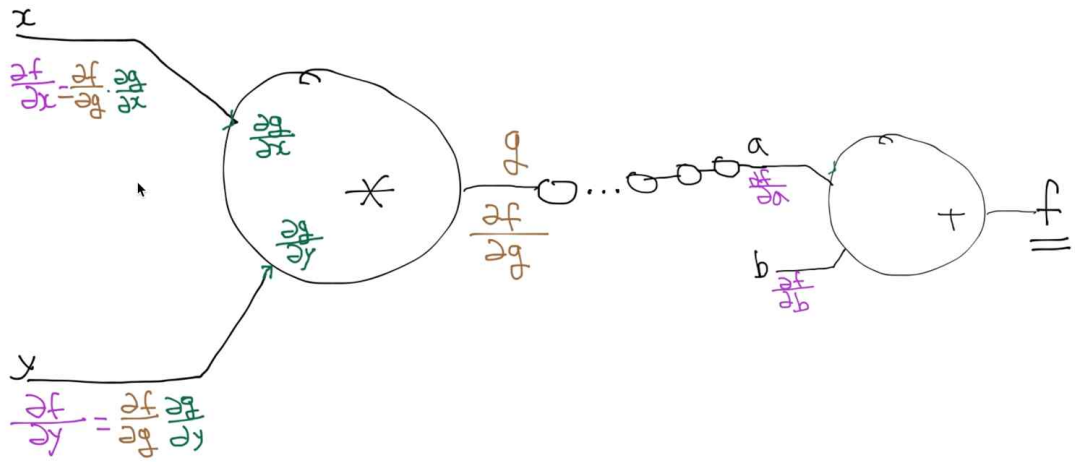


- 실습: AND,_OR,_XOR_연산_신경망_만들기.ipynb

2) Backpropagation(오차역전파)

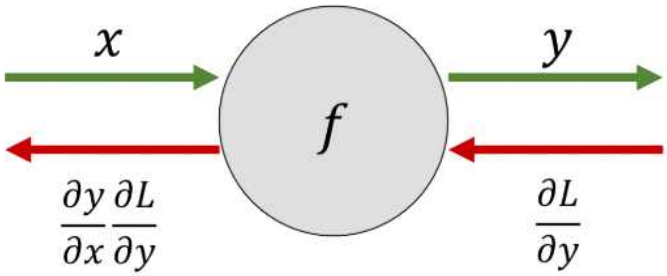






계산그래프와 chain rule)

- 계산그래프(computational graph)는 계산과정을 그래프로 나타낸 것.
- 노드(node, 꼭지점)은 함수(연산), 엣지(edge, 간선)는 값을 뜻한다.
  y=f(x)를 나타내는 계산그래프는 아래 그림과 같다.



- 계산그래프에서 계산을 왼쪽에서 오른쪽으로 진행하는 단계를 순전파(forward propagation)라고 하고, 입력값 x는 함수 f를 거쳐 y로 순전파되고 있다.
- 반대로 계산을 오른쪽에서 왼쪽으로 진행하는 단계를 역전파(backward propagation)
- ∂L/∂y의 의미: 우리의 목적은 뉴럴네트워크의 오차를 줄이는 데 있기 때문에, 각 파라메터별로 Loss에 대한 그래디언트를 구한 뒤 그래디언트들이 향한 쪽으로 파라메터들을 업데이트한다. **∂L/∂y는 y에 대한 Loss의 변화량**, 즉 **Loss로부터 흘러들어온 그래디언트를 뜻함**
- ∂L/∂x : 현재 입력값 x에 대한 Loss의 변화량, 미분의 연쇄법칙(chain rule)계산식

$$\frac{\partial L}{\partial x} = \frac{\partial y}{\partial x}\frac{\partial L}{\partial y}$$

- ∂L/∂y는 Loss로부터 흘러들어온 그래디언트
- ∂y/∂x는 현재 입력값에 대한 현재 연산결과의 변화량, 즉 로컬 그래디언트(Local Gradient)
- **현재 입력값에 대한 Loss의 변화량은 Loss로부터 흘러들어온 그래디언트에 로컬 그래디언트를 곱해서 구한다는 이야기이고, 이 그래디언트는 다시 앞쪽에 배치돼 있는 노드로 역전파 된다.**

※ Backpropagation 참조 사이트:
https://ratsgo.github.io/deep%20learning/2017/05/14/backprop/

Softmax-with-Loss 노드)

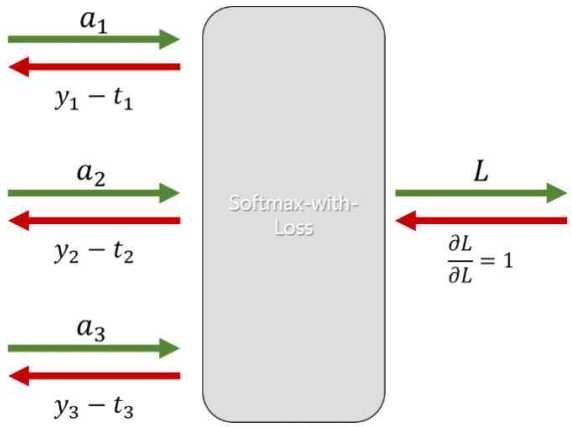
- 뉴럴네트워크 말단에 보통 Softmax-with-Loss 노드를 둔다.
- Softmax-with-Loss란 소프트맥스 함수와 교차 엔트로피(Cross-Entropy) 오차를 조합한 노드를 뜻한다.
- 소프트맥스 함수와 교차 엔트로피의 수식

$$y_k = \frac{exp(a_k)}{\sum_{i=1}^{n} exp(a_i)}$$

$$L = -\sum_{k} t_k \log y_k$$

- ak=노드의 입력값
- L=노드의 출력값(Loss)
- tk=정답 레이블(0 혹은 1)
- n=정답 범주 개수

- Softmax-with-Loss 노드의 계산그래프



- Softmax-with-Loss 노드는 a를 입력으로 받아서 Loss L을 출력
- 역전파하는 그래디언트는 yk−tk, 정답이 t3이라면 역전파되는 그래디언트는 각각 y1,y2,y3−1이 된다.
- Softmax-with-Loss 노드의 역전파 그래디언트를 구하려면 입력값에 소프트맥스 확률값을 취한 뒤, 정답 레이블에 해당하는 요소만 1을 빼주면 된다.

```
import numpy as np
p = np.exp(a) / np.sum(np.exp(a)) # softmax 확률 계산
da = np.copy(p)
da[target] -= 1 # target=정답 인덱스를 갖고 있는 변수
```

(softmax 파이썬 코드)

◑ 미분 ◑

Basic derivative

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

f를 x로 미분	x가 0으로 순간 변화률

미분계산)

f(x)=0 ; 0

f(x)=x ; 1

f(x)=2x ; 2

$$\frac{d}{dx}f(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$f(x) = 3$ $\frac{df}{dx} = 0$

$\Delta x = 0.01$ $\frac{f(x+0.01) - f(x)}{0.01} = \frac{3-3}{0.01} = \frac{0}{0.01} = 0$

f(x)=3 ; 0

$f(x) = x$

$\frac{f(x+0.01) - f(x)}{0.01} = \frac{x+0.01-x}{0.01} = \frac{0.01}{0.01} = 1$, $\frac{dx}{dx} = 1$

f(x)=x ; 1

$f(x) = 2x$ $\frac{f(x+0.01) - f(x)}{0.01} = \frac{2(x+0.01) - 2x}{0.01} = \frac{2 \times 0.01}{0.01} = 2$

f(x)=2x ; 2

◑ 편미분 ◑

f(x) = 2x ; 2

f(x,y) = xy, $\frac{\partial f}{\partial x}$; y

f(x,y) = xy, $\frac{\partial f}{\partial y}$; x

$f(x) = 2x$ $\frac{df}{dx} = 2$

$f(x,y) = xy, \frac{\partial f}{\partial x}$; $\frac{\partial f}{\partial x} = y$

$f(x,y) = xy, \frac{\partial f}{\partial y}$; $= x$

$f(x) = 3$ $\frac{df}{dx} = 0$

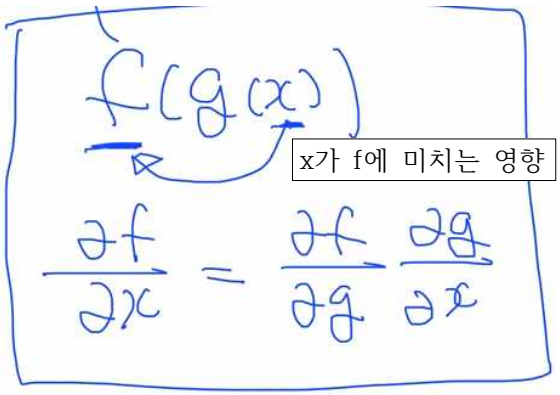
$f(x) = 2x$ $f(x) = x + x$; $\frac{df}{dx} = \frac{dx}{dx} + \frac{dx}{dx} = 2$

$f(x) = x + 3$ $\frac{df}{dx} = 1$

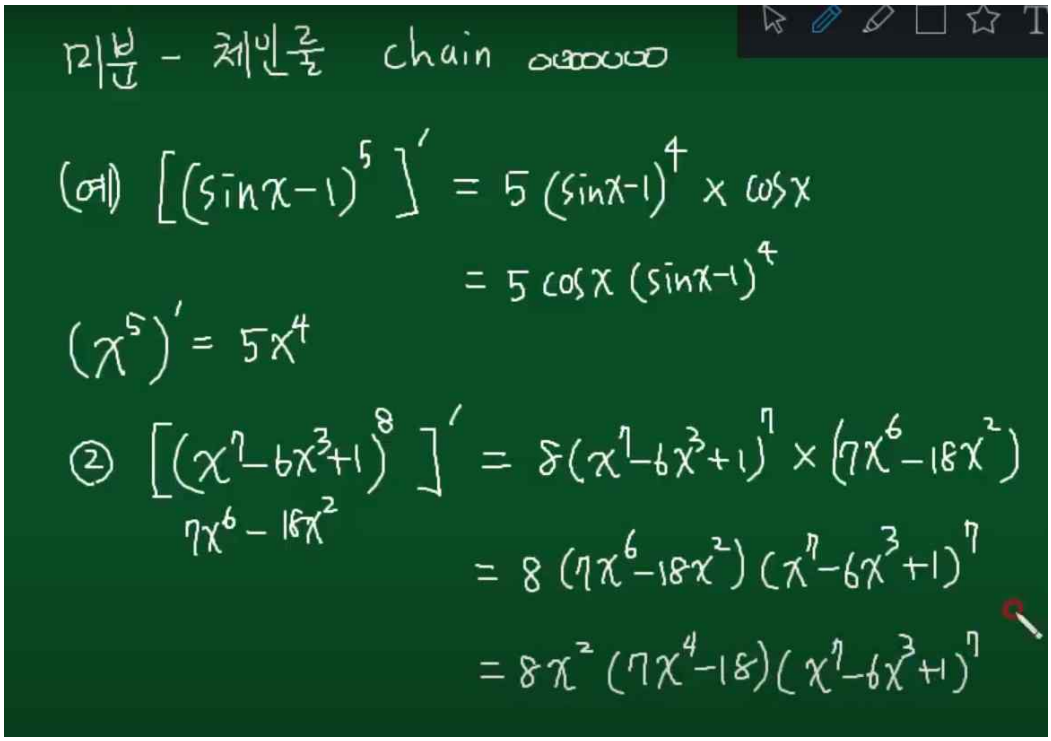
$f(x,y) = x + y, \frac{\partial f}{\partial x} = 1$

$f(x,y) = x + y, \frac{\partial f}{\partial y} = 1$

※ chain rule



참조: https://www.youtube.com/watch?v=bqAOnDOqwbw



③ $[\sin^3(x^2-1)]'$

$\sin^2 x = (\sin x)^2$

$\sin x^2 = \sin(x^2)$

$= [[\sin(x^2-1)]^3]'$

$= 3[\sin(x^2-1)]^2 \times [\sin(x^2-1)]'$

$= 3[\sin(x^2-1)]^2 \times \cos(x^2-1) \times 2x$

$= 6x\cos(x^2-1)\sin^2(x^2-1)$

$* [\sin(x^2-1)]'$

$= \cos(x^2-1) \times 2x$

④ $\left(e^{\sin^2 x - \cos^2 x}\right)' = e^{\sin^2 x - \cos^2 x} \times (\sin^2 x - \cos^2 x)'$

$= e^{\sin^2 x - \cos^2 x} \times (2\sin x\cos x - (-2\cos x \sin x))$

$= e^{\sin^2 x - \cos^2 x}(4\sin x\cos x) = 4\sin x\cos x\, e^{\sin^2 x - \cos^2 x}$

✓ $(\sin^2 x)' = [(\sin x)^2]' = 2(\sin x)' \times \cos x = 2\sin x\cos x$

✓ $(\cos^2 x)' = [(\cos x)^2]' = 2(\cos x)' \times (-\sin x) = -2\cos x \sin x$

[2교시]

1. CNN

1) Convolution의 output 크기 계산하기

$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

예제 1)
input image size : 227 x 227
filter size = 11x11
stride = 4
padding = 0
output image size = ?

예제 2)
input image size : 64 x 64
filter size = 7x7
stride = 2
padding = 0
output image size = ?

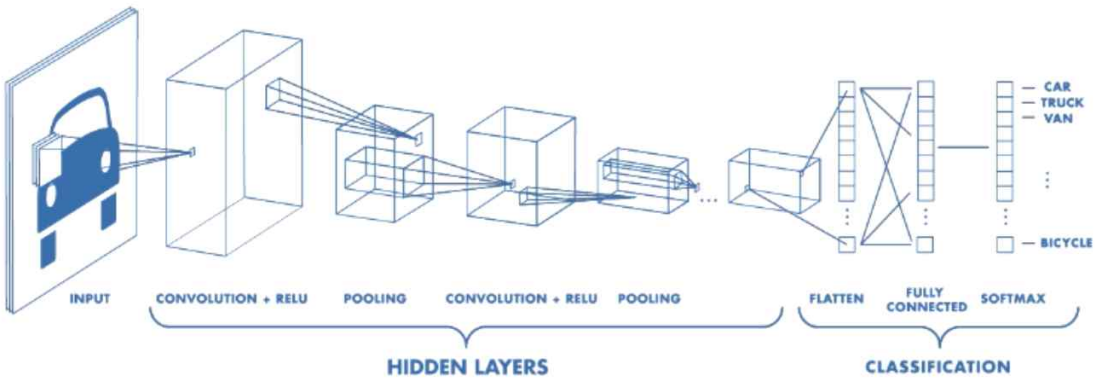
예제 3)
input image size : 32 x 32
filter size = 5x5
stride = 1
padding = 2
output image size = ?

예제 4)
input image size : 32 x 64
filter size = 5x5
stride = 1
padding = 0
output image size = ?

예제 5)
input image size : 64 x 32
filter size = 3x3
stride = 1
padding = 1
output image size = ?

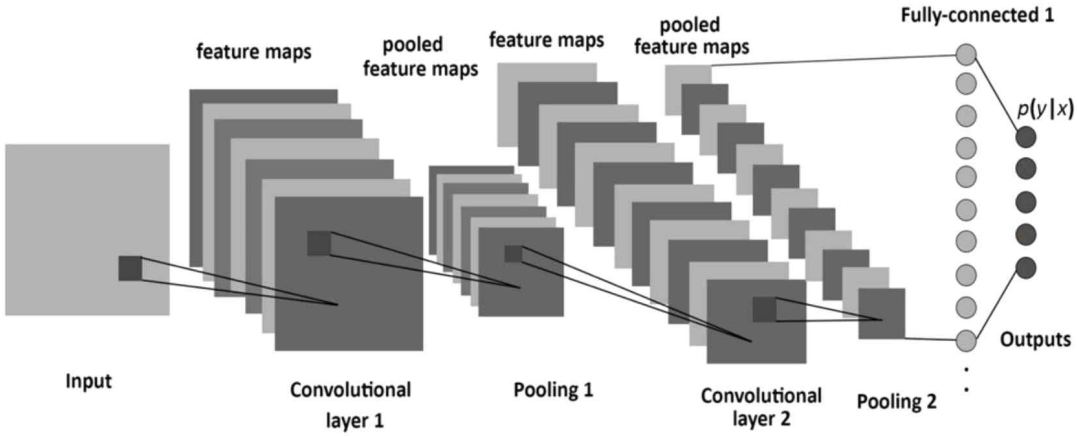
2) CNN주요 모델

- CNN에서 이미지 데이터가 Convolution 연산을 먼저 거치게 될 때 Channel(C)이 커지고, 이미지의 Height(H), Width(W) 가 줄어 듭니다.
- 이 첫번째 연산은 콘볼루션 연산으로 원본 이미지와 함께 연산될 Filter(or Kernel)가 필요한데 이 Filter의 크기가 하이퍼파라미터가 되며 Filter크기에 따라 Computational Resource(계산리소스)가 크게 달라진다.
- Convolution 연산을 위해 Stride와 Padding 값 역시 Hyper Parameter이므로 Filter, Stride, Padding 값에 비례하여 Convolution Layer가 Deep하게 되면 엄청난 연산이 필요하다.



- CNN에서 Convolution Layer에서 연산의 Output 값들을 거의 대부분 Pooling을 거치며 Affine(어파인 공간에서는 벡터에 위치표현을 위한 점을 추가하여 해당 벡터의 크기, 방향 뿐만 아니라 위치까지도 표현할 수 있게 된다)을 자주 사용
- Pooling에서도 Filter Size와 Stride 값 등의 Hyper Parameter도 역시 부담스러운

Computational Resource로 작용함



◑ CNN- Imagenet에 쓰인 주요 모델

ILSVRC 대회

- 과거의 이미지 분류 대회로 현재 이 대회는 공식적으로 종료되었고 캐글에서 대회를 이어가고 있다.
- 대회는 Imagenet이라는 데이터를 사용하는데 1000개의 카테고리와 수백만의 이미지 데이터로 이루어져 있다.

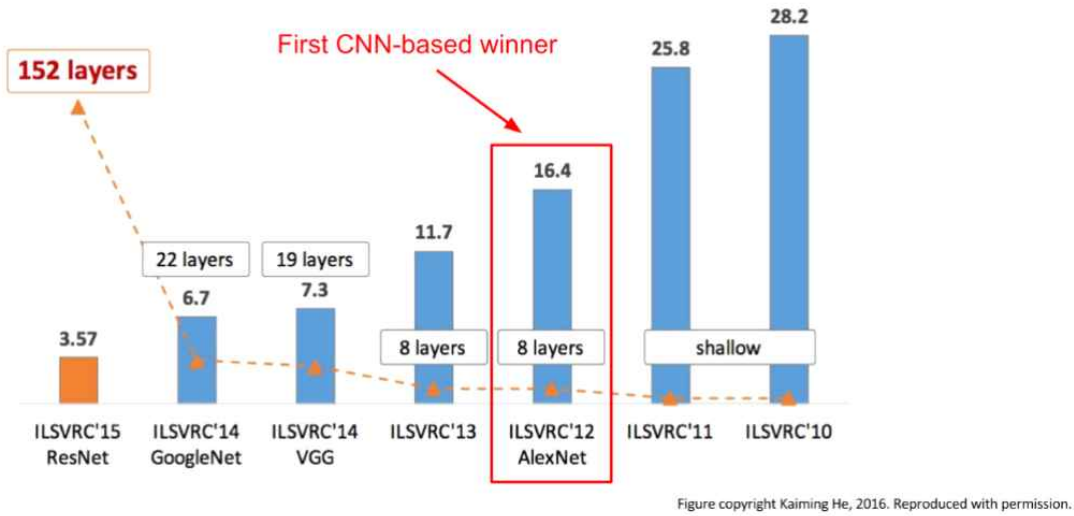
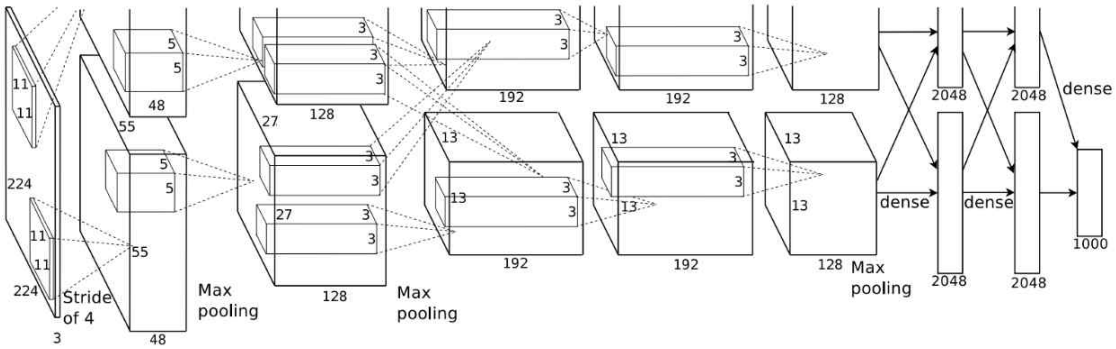


Figure copyright Kaiming He, 2016. Reproduced with permission.

가) AlexNet

CNN은 기본적으로 얀 르쿤이 1989년 제안한 구조를 토대로 하고 있다. 컴퓨터 비전 분야의 '올림픽'이라 할 수 있는 ILSVRC(ImageNet Large-Scale Visual Recognition Challenge)의 2012년 대회에서 제프리 힌튼 교수팀의 AlexNet이 top 5 test error 기준 15.4%를 기록해 2위(26.2%)를 큰 폭으로 따돌리고 1위를 차지

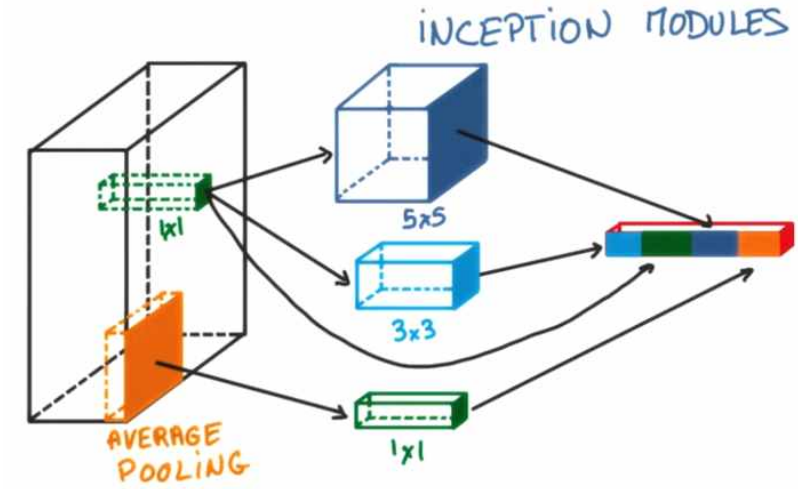


- conv layer, max-pooling layer, dropout layer 5개
- fully connected layer 3개
- nonlinearity function : ReLU
- batch stochastic gradient descent

- AlexNet이 중요한 이유는 의미 있는 성능을 낸 첫번째 CNN 아키텍처이자, AlexNet에 쓰인 드롭아웃 등 기법은 이 분야 표준으로 자리 잡을 정도로 선도적인 역할을 했기 때문

나) GoogleNet

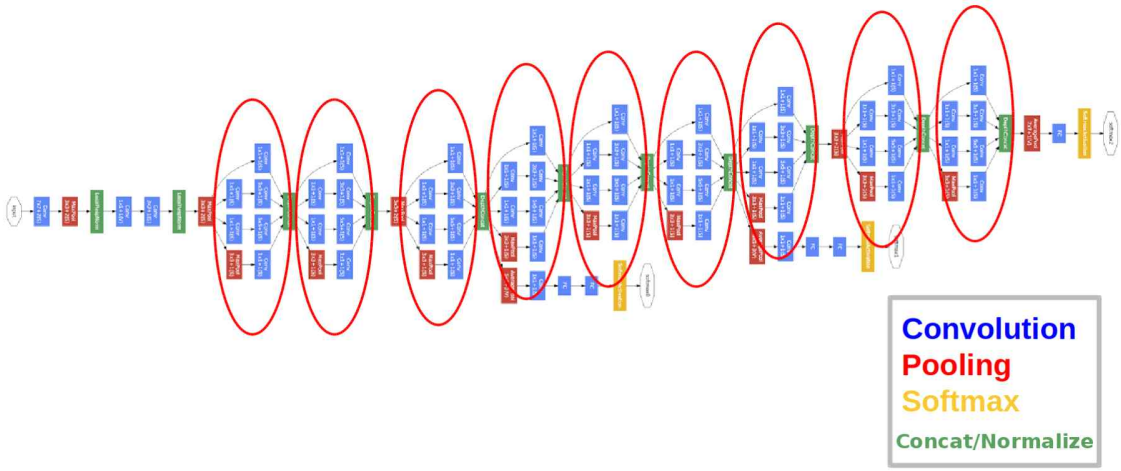
- AlexNet 이후 층을 더 깊게 쌓아 성능을 높이려는 시도들이 계속되었습니다. VGGNet(2014), GoogleNet(2015) 등이 바로 그것
- GoogleNet은 VGGNet보다 구조가 복잡해 널리 쓰이진 않았지만 아키텍처 면에서 주목을 받았다.
- 보통 하나의 conv layer에는 한 가지의 conv filter가 사용됨
- 하나의 layer에서 다양한 종류의 filter나 pooling을 도입함으로써 개별 layer를 두텁게 확장시킬 수 있다는 Inception module을 제안함



- Inception module에서 특히 주목받은 것이 바로 1×1 conv filter이다.
- 가령 현재 층 입력데이터 이미지의 차원수가 100×100×60이고, 1×1 conv filter를 20개 사용한다면 데이터의 차원 수는 100×100×20으로 줄어든다. 60개 채널(차원)에 달하는 하나의 픽셀이 20차원의 feature space로 선형변환, 차원축소된 것이라고도 볼 수 있겠다.

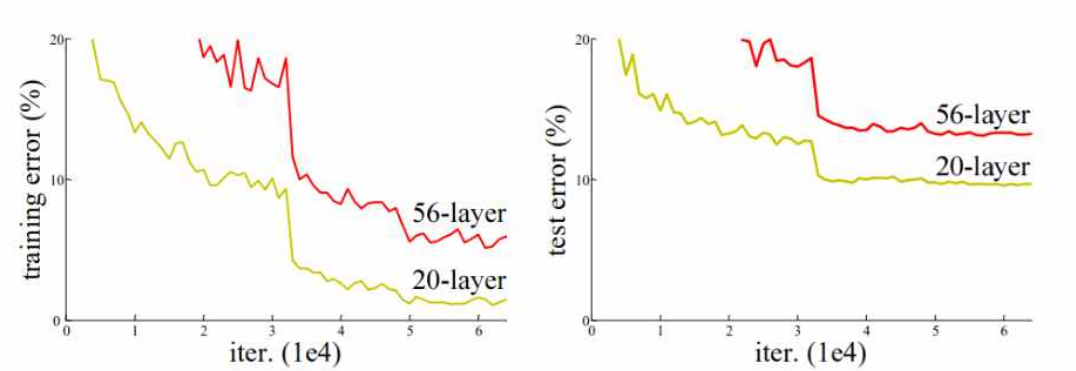
※ Inception 내용 참조

https://ikkison.tistory.com/86



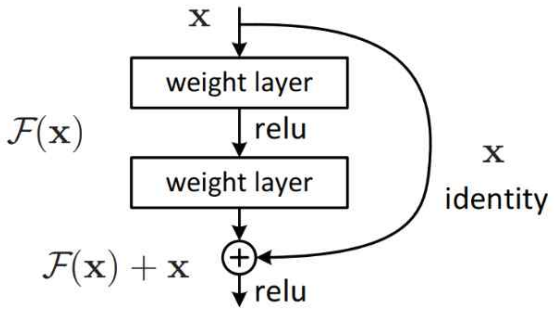
다) ResNet

- ResNet(2015)은 2015년 ILSVRC에서 오류율 3.6%로 1등을 차지
- AlexNet이 처음 제안된 이후로 CNN 아키텍처의 층은 점점 더 깊어졌다. AlexNet이 불과 5개 층에 불과한 반면 VGGNet은 19개 층, GoogleNet은 22개 층에 달한다.
- 하지만 층이 깊어질수록 역전파되는 그래디언트가 중간에 죽어서 학습이 잘 되지 않는 문제(gradient vanishing)가 발생한다.



- ResNet의 핵심 아이디어는 residual block이다.

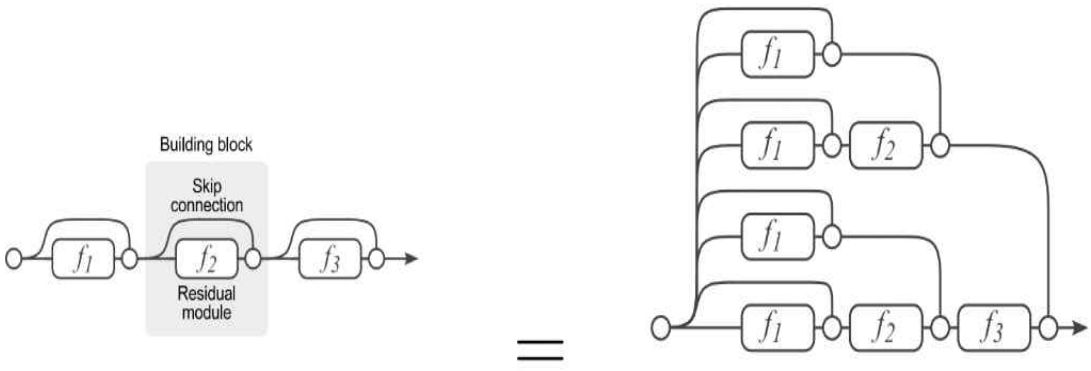
그래디언트가 잘 흐를 수 있도록 일종의 지름길(shortcut, skip connection)을 만들어 주자는 생각으로, 이는 forget gate 등을 도입해 이전 스텝의 그래디언트(정보)를 좀 더 잘 흐르게 만드려는 Long Term Short Memory(LSTM)의 철학과 본질적으로 유사함.



- ResNet의 성능이 좋은 이유는 그래디언트 문제의 해결 말고도 residual block이 앙상블(ensemble) 모델을 구축한 것과 비슷한 효과를 낸다고 함

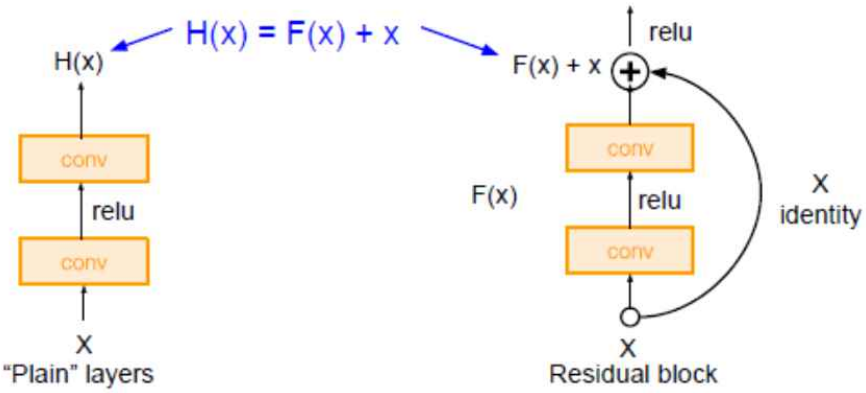
※ 앙상블; 여러 개의 모델을 조화롭게 학습시켜 그 모델들의 **예측 결과들을 이용하여 더 정확한 예측값을 구하는 것**

- residual block의 skip connection 덕분에 입력데이터와 그래디언트가 오갈 수 있는 통로가 크게 늘어나기 난다.(n개 skip connection이 있다면 2n개의 서로 다른 통로 존재)



(a) Conventional 3-block residual network (b) Unraveled view of (a)

※ skip connection



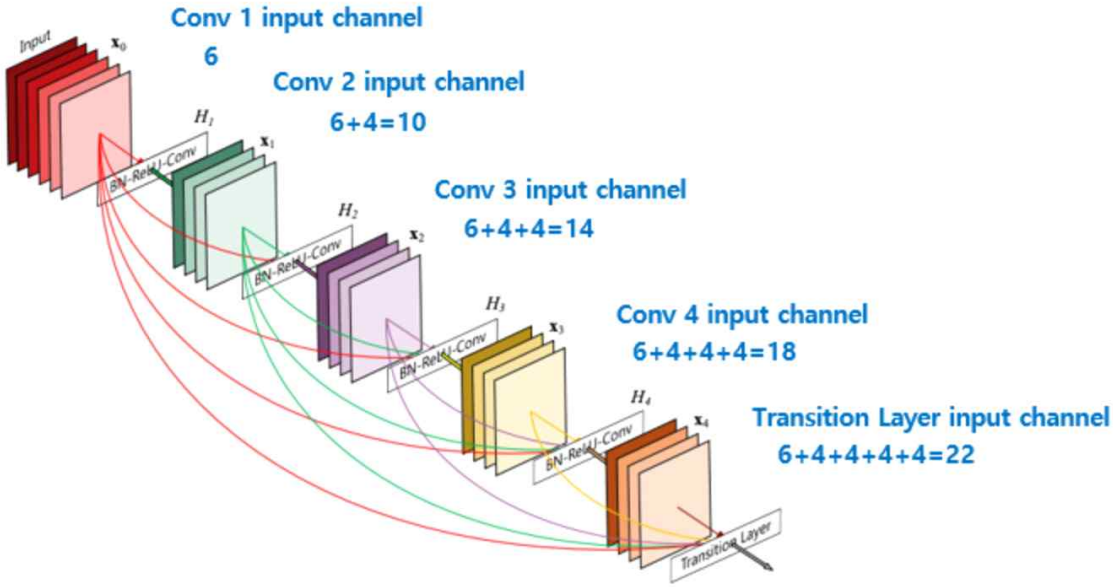
(Plane layer) (Residual Block)

- 동일한 연산을 하고 나서 Input x를 더하는 것(Residual Block)과 더하지 않는 것(Plane layer)이 차이점

- 단순하게 Input x를 더하는 것만으로 레이어는 Direct로 학습하는 것 대신에, Skip Connection을 통해 각각의 Layer(Block)들이 작은 정보들을 추가적으로 학습하도록 한다(= 각각의 레이어가 배워야 할 정보량을 축소)

라) DenseNet

- DenseNet(2016)은 ResNet에서 한발 더 나아가 전체 네트워크의 모든 층과 통하는 지름길을 만들었다.



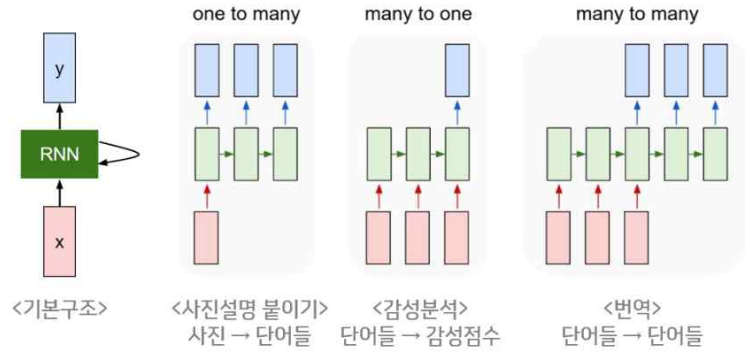
- 이전 layer들의 feature map을 계속해서 다음 layer의 입력과 연결하는 방식이며 이러한 방식은 ResNet에서도 사용이 되었다.
- 다만 ResNet은 feature map 끼리 더하기를 해주는 방식이었다면 DenseNet은 feature map끼리 Concatenation 을 시키는 것이 가장 큰 차이점
- 이러한 구조가 얻을 수 있는 이점은

• Vanishing Gradient 개선
• Feature Propagation 강화
• Feature Reuse(재사용)
• Parameter 수 절약

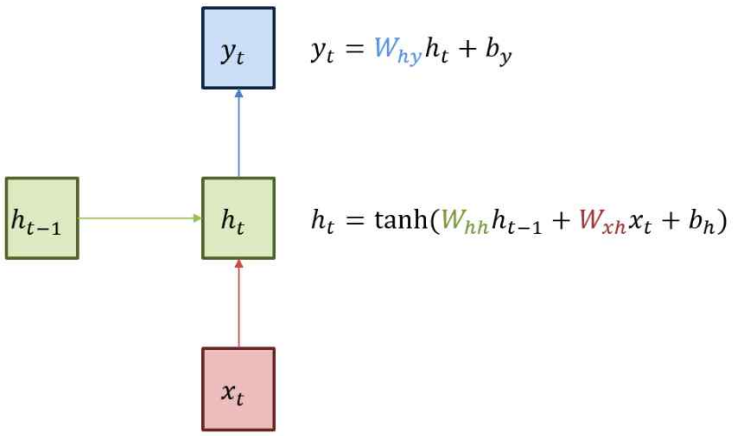
[3교시]

1. RNN

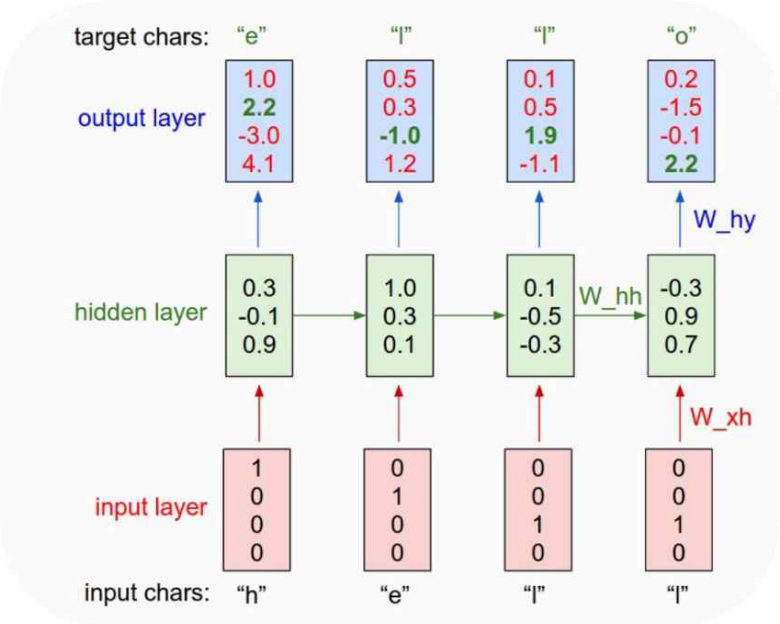
RNN의 기본 구조



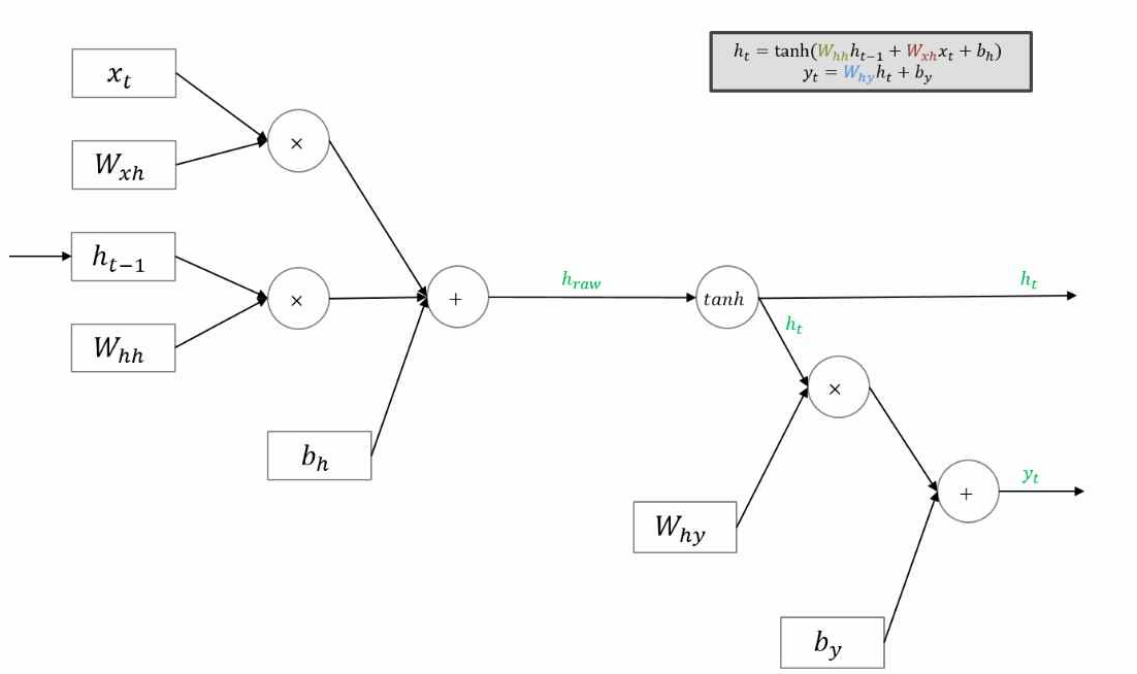
- RNN은 히든 노드가 방향을 가진 엣지로 연결돼 순환구조를 이루는(directed cycle) 인공신경망의 한 종류
- 음성, 문자 등 순차적으로 등장하는 데이터 처리에 적합한 모델



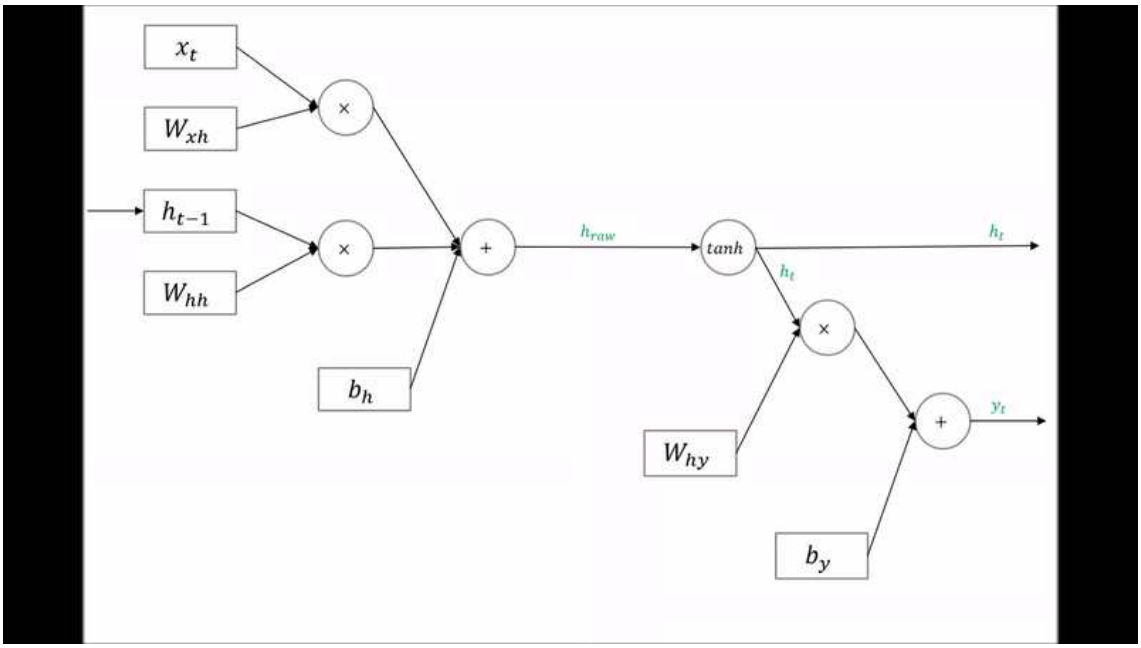
- RNN의 기본 구조
- 녹색 박스는 히든 state
- 빨간 박스는 인풋 x
- 파란 박스는 아웃풋 y
- 현재 상태의 히든 state ht는 직전 시점의 히든 state ht−1를 받아 갱신
- 현재 상태의 아웃풋 yt는 ht를 전달받아 갱신되는 구조
- 히든 state의 활성함수(activation function)은 비선형 함수인 하이퍼볼릭탄젠트(tanh)



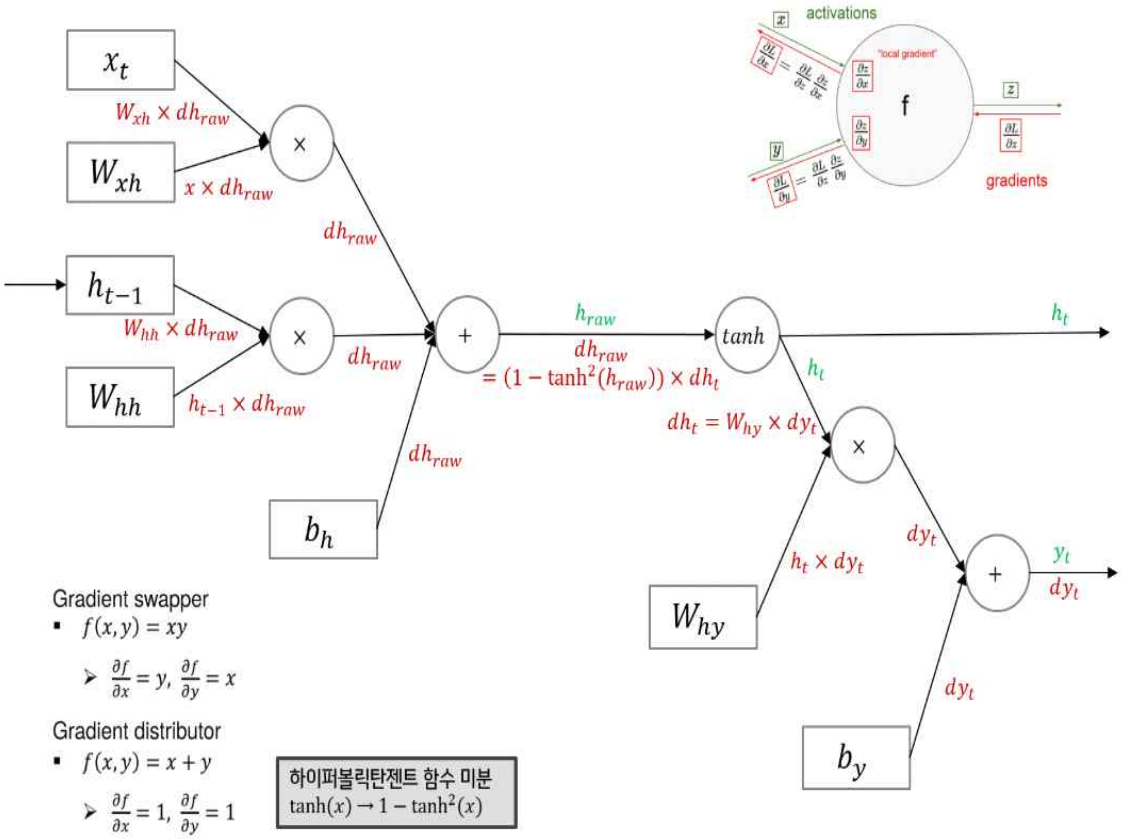
- x1은 [1,0,0,0]
- 이를 기반으로 h1인 [0.3,−0.1,0.9]를 만든다(h0는 존재하지 않기 때문에 랜덤 값을 집어넣는다).
- 이를 바탕으로 y1인 [1.0,2.2,−3.0,4.1]로 생성
- 마찬가지로 두번째, 세번째, 네번째 단계들도 모두 갱신한다(순전파(foward propagation))
- 모델에 정답을 알려줘야 parameter를 적절히 갱신해 나간다.
- 바로 다음 글자가 정답이 되는 구조, 'h'의 다음 정답은 'e', 'e' 다음은 'l', 'l' 다음은 'l', 'l' 다음은 'o'가 정답이다.
- 첫번째 정답인 'e'는 두번째 요소만 1이고 나머지가 0인 one-hot-vector
- 아웃풋에 진한 녹색으로 표시된 숫자: 정답에 해당하는 인덱스를 의미
- 이 정보를 바탕으로 역전파(backpropagation)를 수행해 parameter값들을 갱신
- parameter: 인풋 x를 히든레이어 h로 보내는 Wxh, 이전 히든레이어 h에서 다음 히든레이어 h로 보내는 Whh, 히든레이어 h에서 아웃풋 y로 보내는 Why
- 모든 시점의 state에서 이 parameter는 동일하게 적용(shared weights).



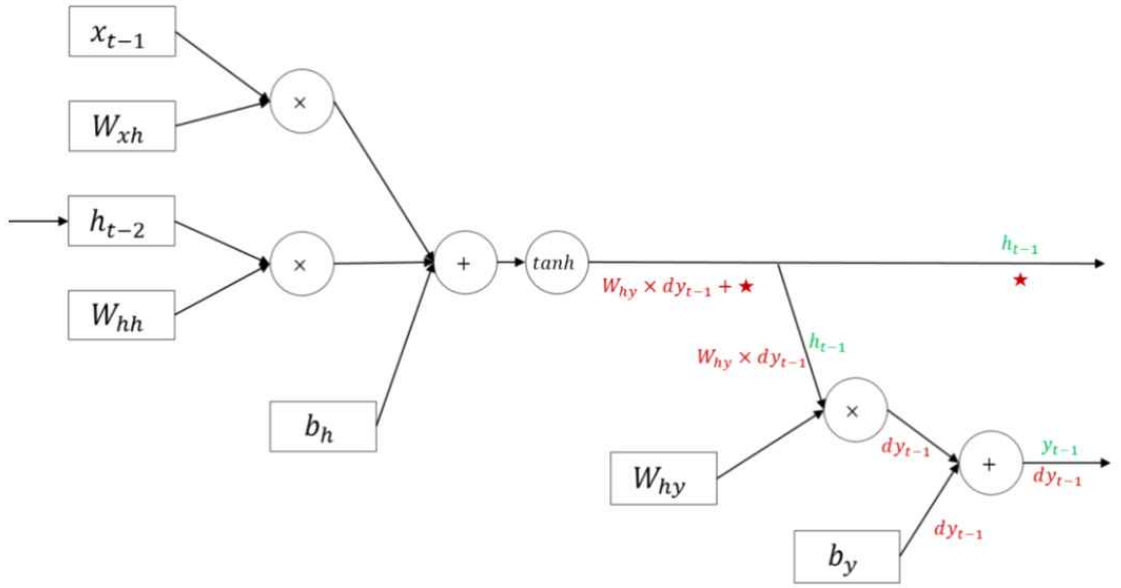
RNN의 역전파)



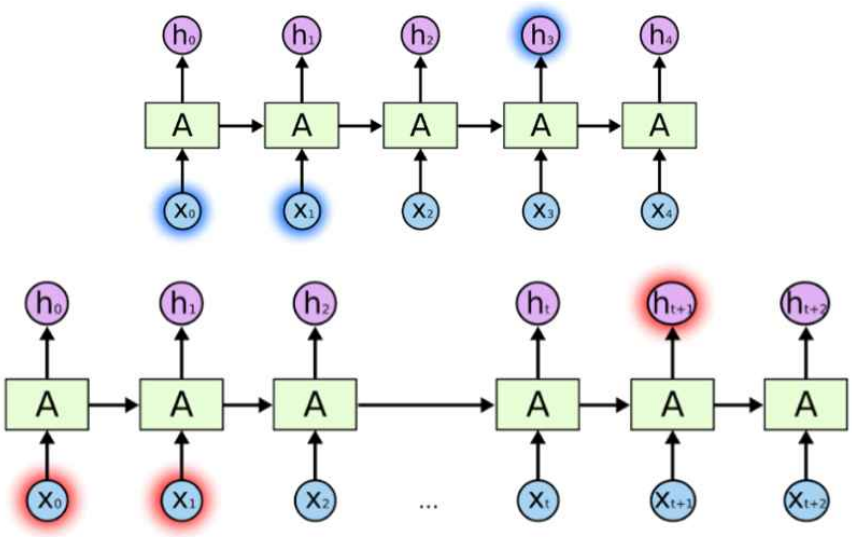
- forward pass를 따라 최종 출력되는 결과는 yt
- 최종 Loss에 대한 yt의 그래디언트(dL/dyt)가 RNN의 역전파 연산에서 가장 먼저 등장, dyt라고 표기, 순전파 결과 yt와 대비해 붉은색으로 표시,
- dyt는 덧셈 그래프를 타고 양방향에 모두 그대로 분배
- dWhy는 흘러들어온 그래디언트 dyt에 로컬 그래디언트 ht를 곱해 구한다.
- dht는 흘러들어온 그래디언트 dyt에 Why를 곱한 값
- dhraw는 흘러들어온 그래디언트인 dht에 로컬 그래디언트인 1−tanh2(hraw)을 곱해 구한다. 나머지도 동일한 방식으로 구하면 된다.



- RNN은 히든 노드가 순환 구조를 띄는 신경망이다. 즉 ht를 만들 때 ht−1가 반영된다. 즉, 아래 그림의 dht−1은 t-1 시점의 Loss에서 흘러들어온 그래디언트인 Why∗dyt−1뿐 아니라 ★에 해당하는 그래디언트 또한 더해져 동시에 반영된다는 뜻

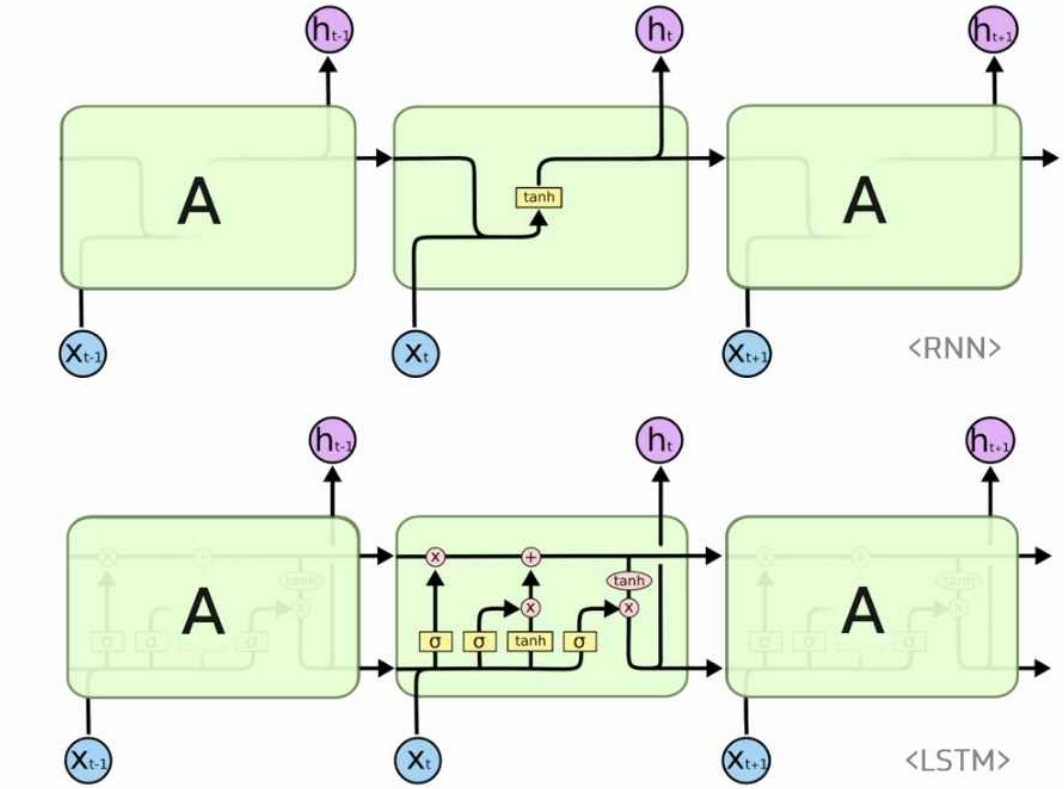


[LSTM의 기본 구조]



<관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 RNN 학습능력 저하>

- RNN은 관련 정보와 그 정보를 사용하는 지점 사이 거리가 멀 경우 역전파시 그래디언트가 점차 줄어 학습능력이 크게 저하되는 **vanishing gradient problem**를 가지고 있다.
- 이 문제를 극복하기 위해서 고안된 것이 바로 LSTM이다.
- **LSTM은 RNN의 히든 state에 cell-state를 추가**한 구조

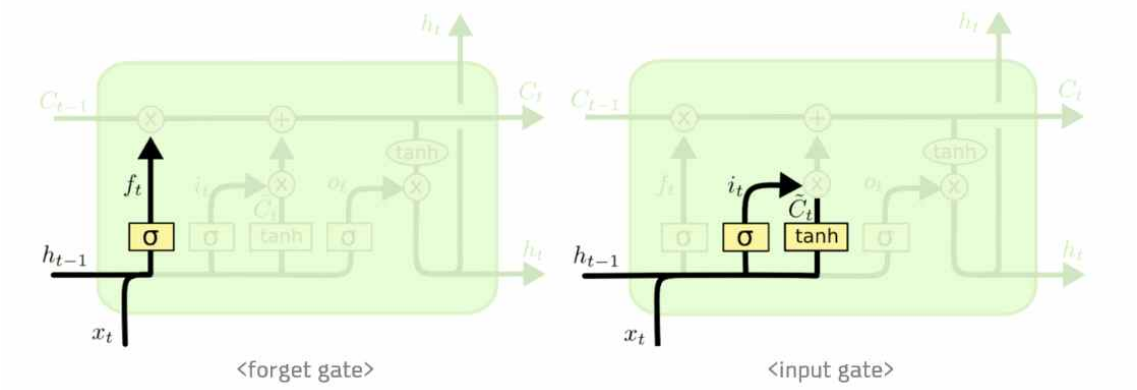


- cell state는 일종의 컨베이어 벨트 역할을 한다.
- 덕분에 state가 꽤 오래 경과하더라도 그래디언트가 비교적 전파가 잘 된다.
- LSTM 셀의 수식

$$
\begin{aligned}
f_t &= \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f}) \\
i_t &= \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i}) \\
o_t &= \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o}) \\
g_t &= \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

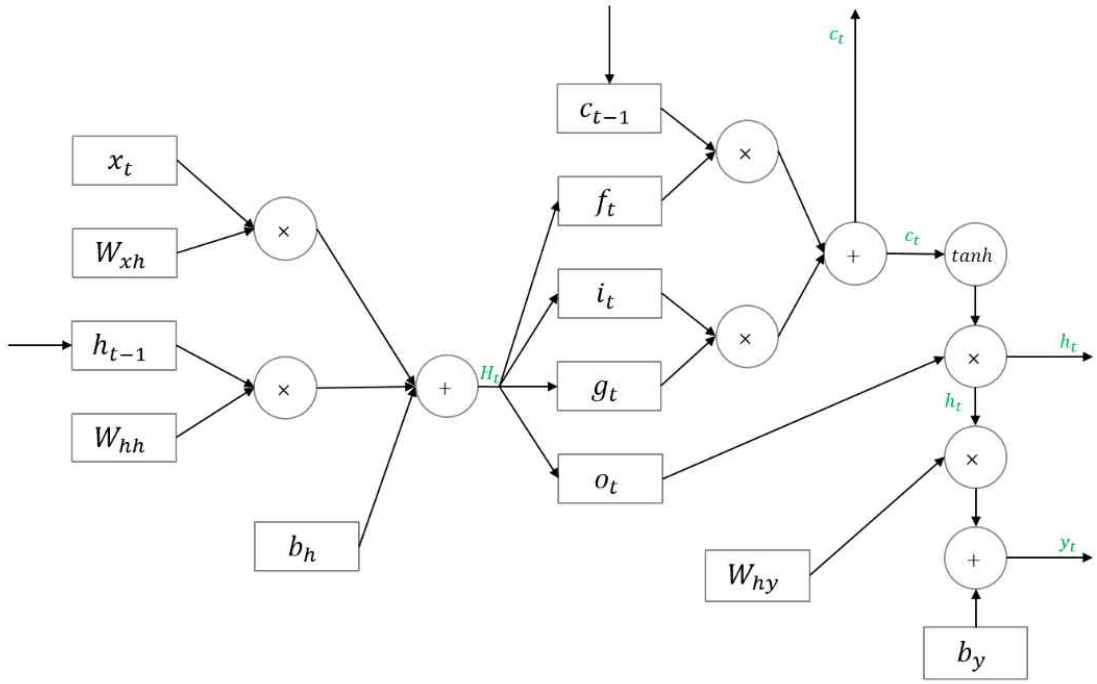
• forget gate ft: 과거 정보를 잊기를 위한 게이트
• ht−1과 xt를 받아 시그모이드를 취해준 값이 바로 forget gate가 내보내는 값
• 시그모이드 함수의 출력 범위는 0에서 1 사이이기 때문에 그 값이 0이라면 이전 상태의 정보는 잊고, 1이라면 이전 상태의 정보를 온전히 기억하게 된다.

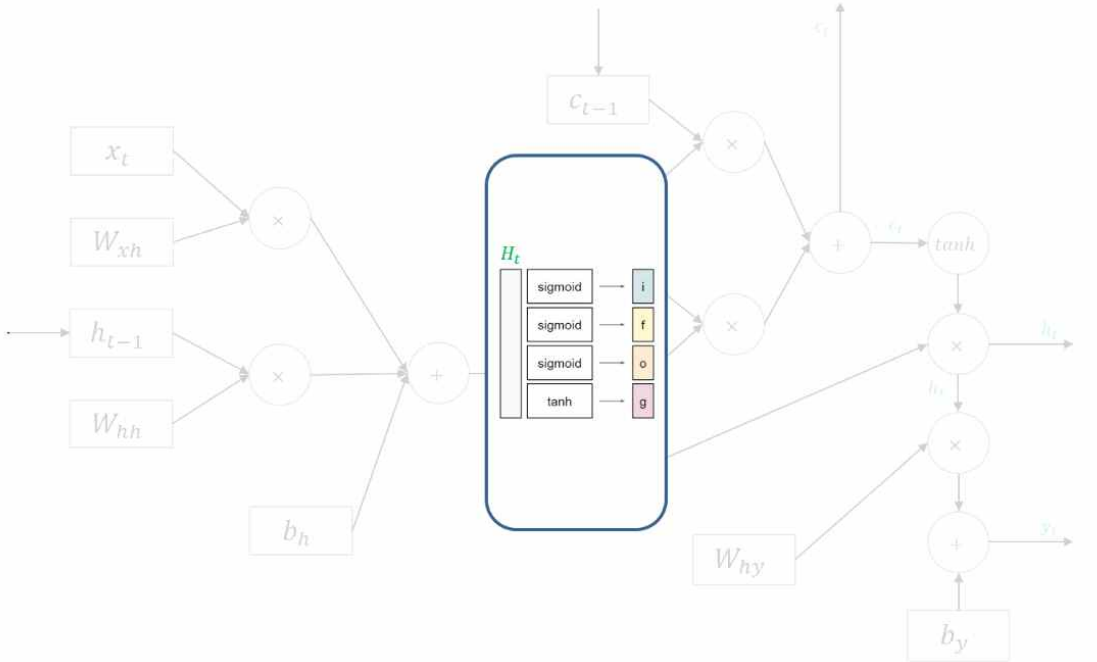
- input gate it⊙gt는 '현재 정보를 기억하기' 위한 게이트
- ht−1과 xt를 받아 시그모이드를 취하고, 또 같은 입력으로 하이퍼볼릭탄젠트를 취해준 다음 Hadamard product 연산을 한 값이 바로 input gate가 내보내는 값이 된다.
- it의 범위는 0~1(강도), gt의 범위는 -1~1(방향)



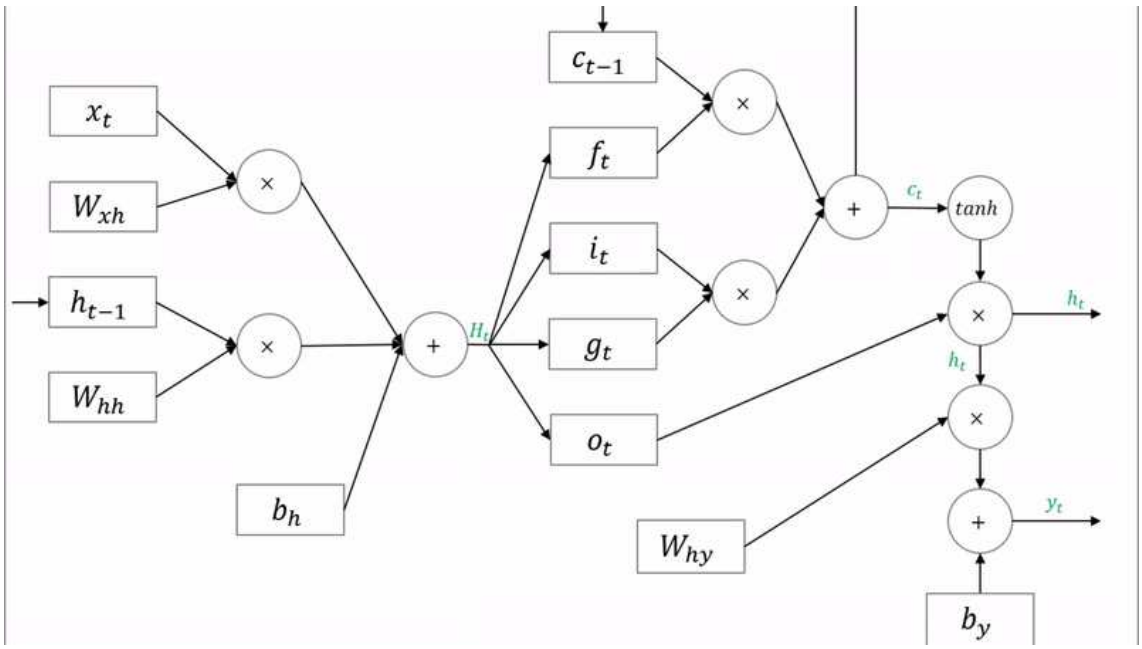
LSTM의 순전파)

- Ht는 행렬을 행 기준으로 4등분해 i,f,o,g 각각에 해당하는 활성함수를 적용하는 방식으로 i,f,o,g를 계산한다.

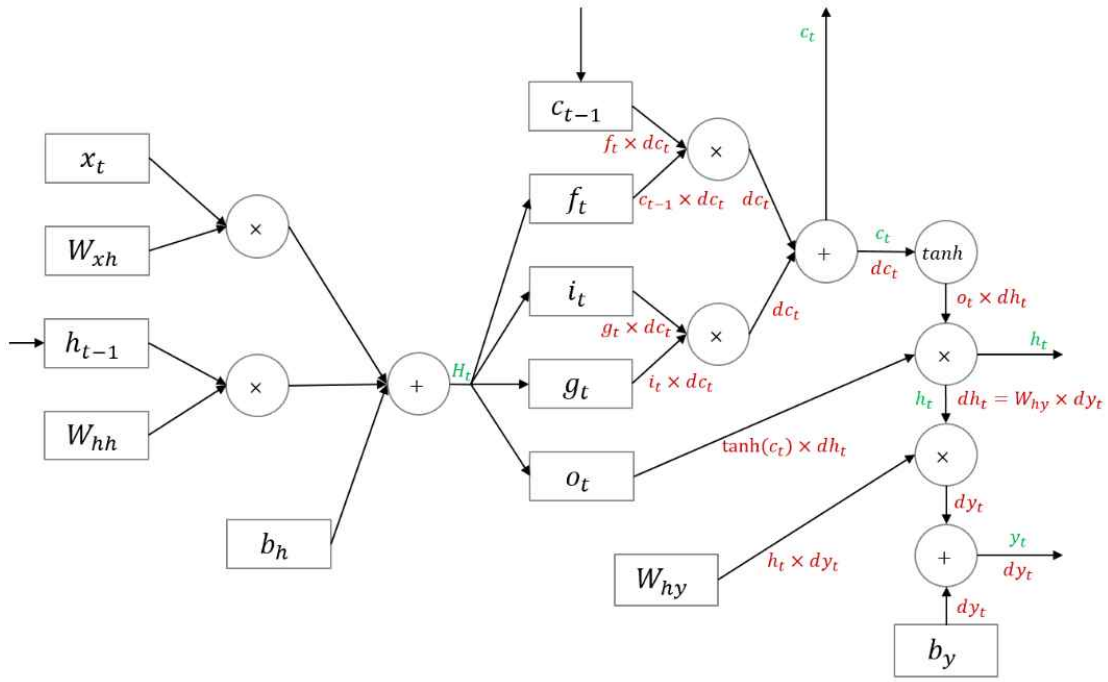




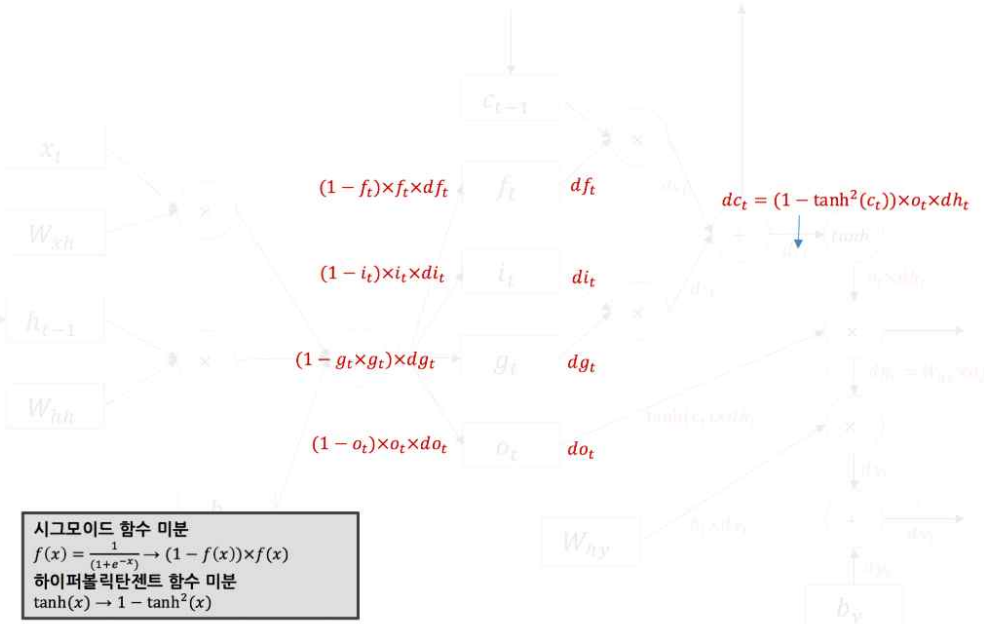
LSTM의 역전파)



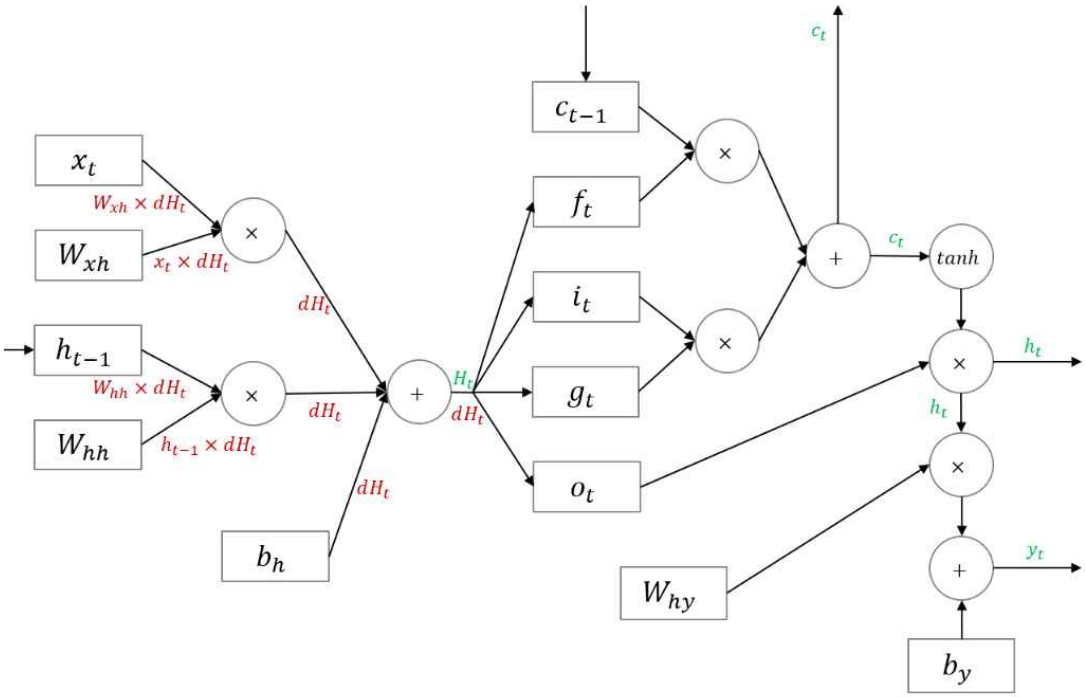
- dft,dit,dgt,dot를 구하기까지 backward pass는 RNN과 유사



- dHt를 구하는 과정이 LSTM backward pass 핵심
- Ht는 it,ft,ot,gt로 구성된 행렬이다. 각각에 해당하는 그래디언트를 이를 합치면(merge) dHt를 만들 수 있다.
- i,f,o의 활성함수는 시그모이드이고, g만 하이퍼볼릭탄젠트이다.
- 각각의 활성함수에 대한 로컬 그래디언트를 구해 흘러들어온 그래디언트를 곱해주면 된다.



- 순전파 과정에서 Ht를 4등분해 it,ft,ot,gt를 구했던 것처럼, backward pass에서는 di,df,do,dg를 다시 합쳐 dHt를 만든다.
- 이렇게 구한 dHt는 다시 RNN과 같은 방식으로 역전파가 되는 구조



- LSTM은 cell state와 히든 state가 재귀적으로 구해지는 네트워크
- 따라서 cell state의 그래디언트와 히든 state의 그래디언트는 직전 시점의 그래디언트 값에 영향을 받는다. 이는 RNN과 마찬가지고 이를 역전파시 반영해야 한다.

RNN 실습)

1) RNN에 대하여(08_RNN.ipynb)
2) RNN을 이용한 시계열 데이터 예측; 주식가격과 같은 시계열(time series) 데이터를 예측하는 RNN 모델의 구현
- 참조 사이트: https://excelsior-cjh.tistory.com/184
3) 셰익스피어 텍스트 생성(RNN)
- 참조 사이트: https://www.tensorflow.org/tutorials/text/text_generation
- https://www.youtube.com/watch?v=J1gXBkBfnlc&feature=emb_logo