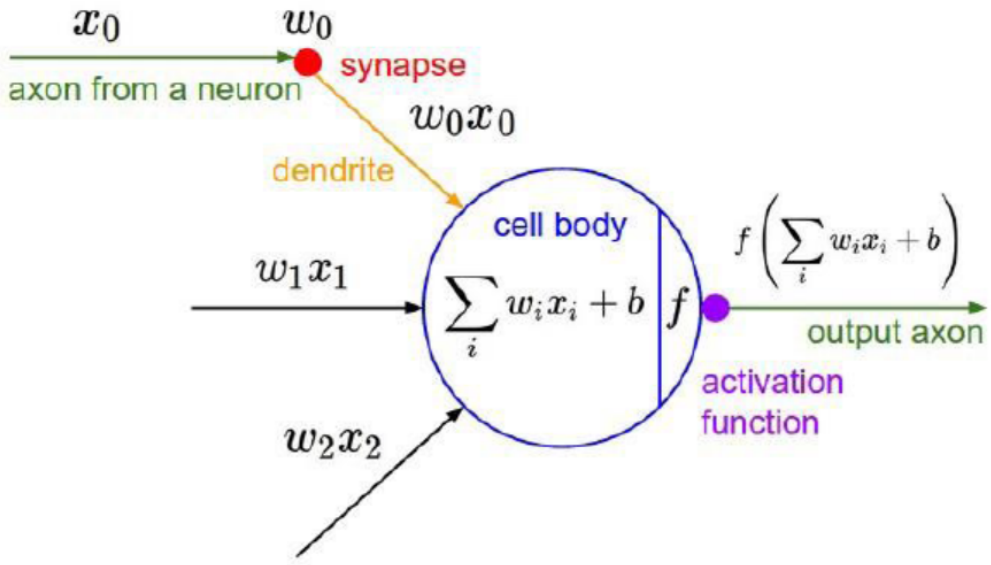
딥러닝 홀로서기#8

1. History of DL / MLP Basic

- 배경 : Machine Learning에서는 한계점이 많았기 때문에 사람처럼 생각하는 모델을 만들어보자는 동기
- 사람의 뉴런구조를 (뉴런 → 전기 자극 → 다음 뉴런) 수학적을 모델링

  입력된 신호 X에 적절한 가중치를 곱하여 다음 뉴런으로 넘김
- 뉴런은 특정 역치값을 넘어야 신호를 처리하는 성질을 가지고 있기 때문에 이를 Activation Function으로 구현 (Non-Linear함)
- Linear Regression이랑은 본질적으로 다르지 않음

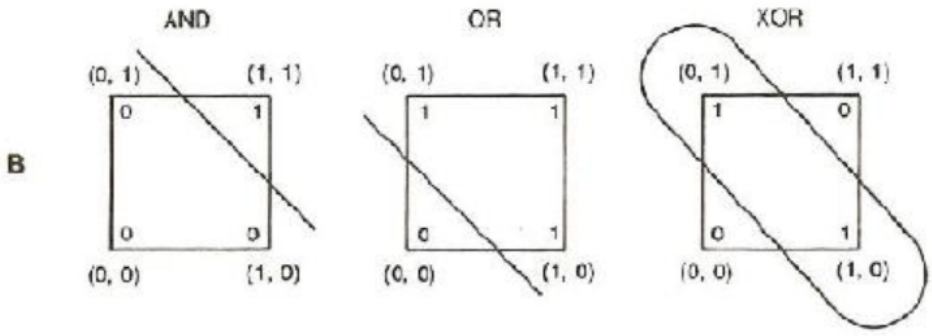


- 간단한 구조의 인공신경망

  $Input1(X_1)$과 $Input(X_2)$를 입력 후 가중치 $W1, W2$를 적용하여 Output 출력

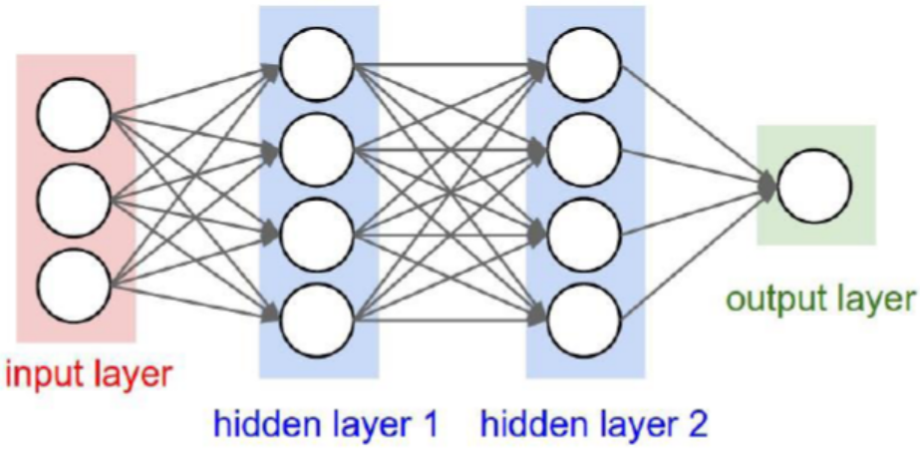
$$Output = W_1X_1 + W_2X_2$$

- 이를 활용하여 AND, OR를 분류하여 보자
  - AND, OR의 경우 두 가지 Input을 받아서 Output은 0과 1중의 하나를 출력하는 Binary Classification문제와 같다. 그래서 간단히 분류가 가능하다



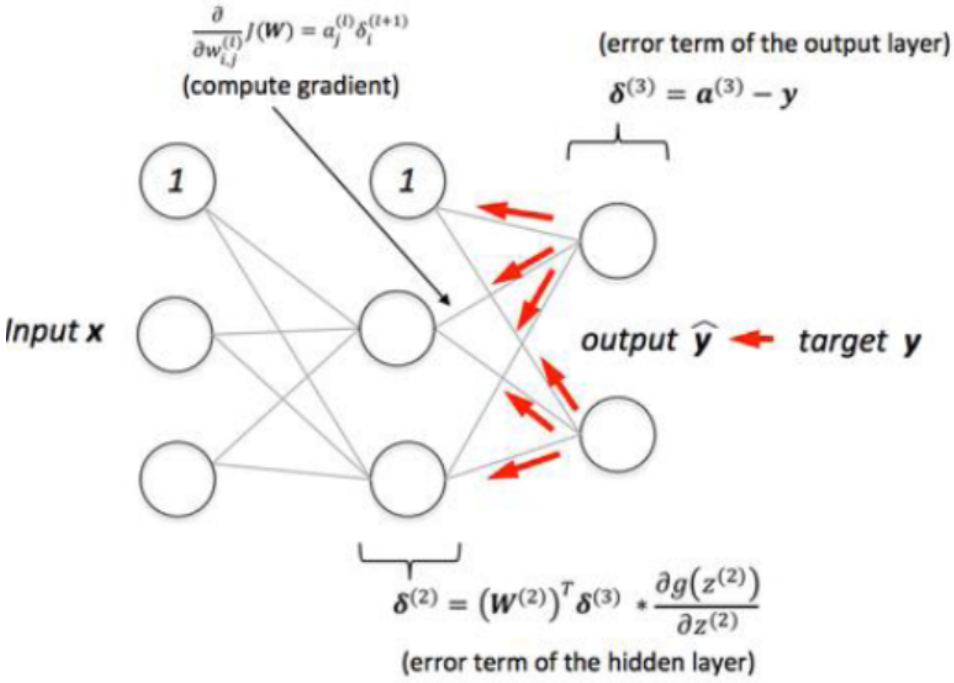
- Mutilayer Perceptron의 등장 (1969년)

  중간에 (Output Layer를 제외) Layer가 하나만 있는 모델로는(=Perceptron Model) 절대 XOR 문제를 풀 수 없다는 것을 MIT의 Marvin Minsky교수가 밝혀냄
- Multilayer Perceptron (Output Layer를 제외하고 중간층에 Hidden Layer가 존재하는 모델)으로 이 문제를 풀 수는 있지만, 복잡한 각각의 Weight의 Bias들을 처리하기가 어려워 Train이 어렵다는 결론을 내림 (이후 20년동안 AI연구의 침체기)



- Backpropagation 알고리즘의 등장 (1986년)

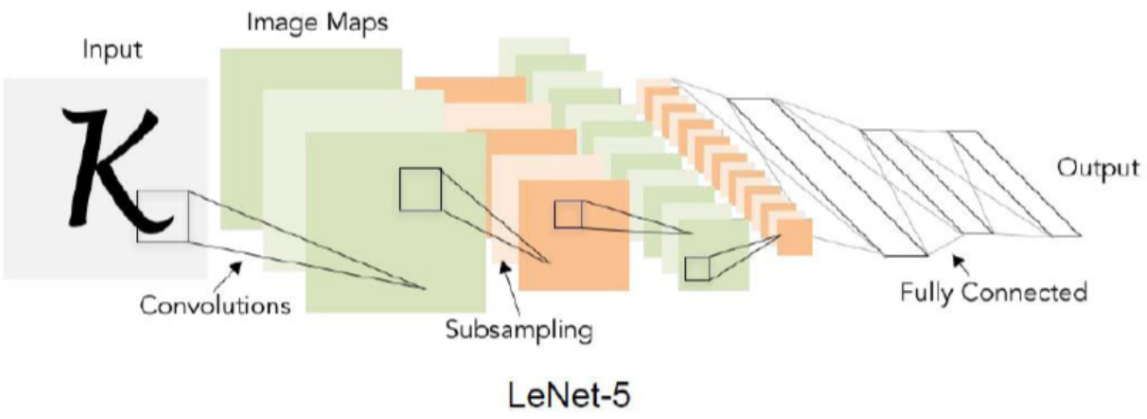
임의의 Output Y (=Predicted Y)와 Target Y와의 Loss를 계산 한 후 이를 바탕으로 앞 선 뉴런의 Gradient를 계산하기를 반복하여(거꾸로 계산) 모든 뉴런의 파라메터에 대한 Gradient를 계산하여 업데이트 하는 방식



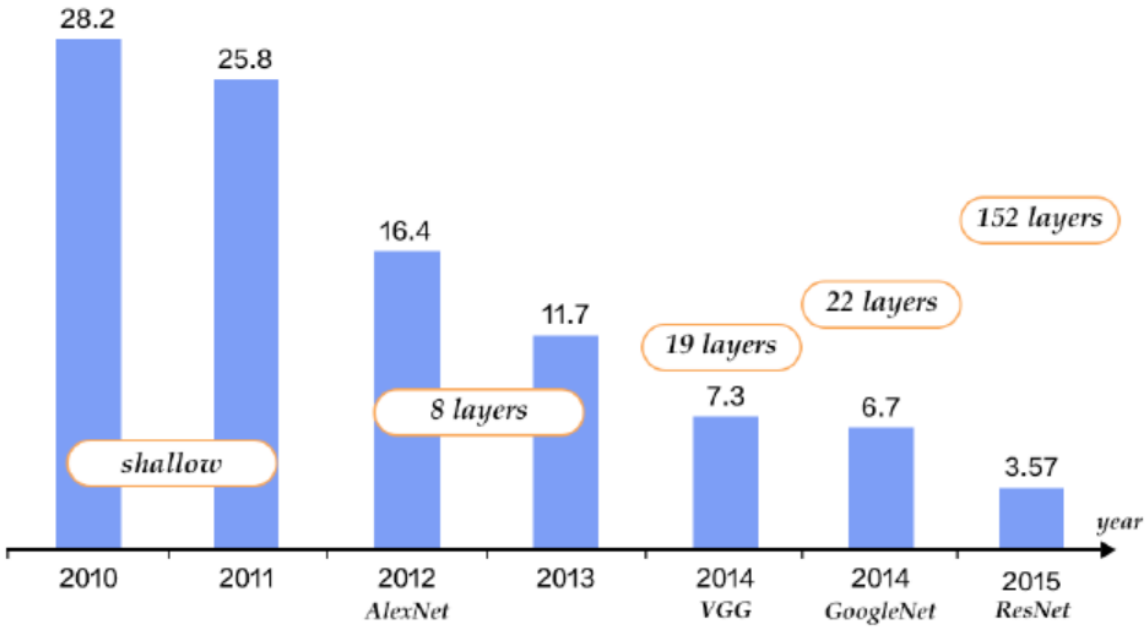
- Convolutional Neural Networks (2012년)

  고양이의 뇌를 관찰한 결과, 이미지의 모양에 따라 활성화되는 뉴런이 다르다는 것을 밝혀냄

  이를 바탕으로 하나의 뉴런이 조그만 이미지를 인식하고, 각각의 뉴런이 합쳐져 전체 이미지를 만든다는 개념의 Conveolutional Neural Networks를 개발



- 점점 Neural Networks가 발전하여 이제는 이미지 분류의 오차율이 3.5%까지 좁혀짐

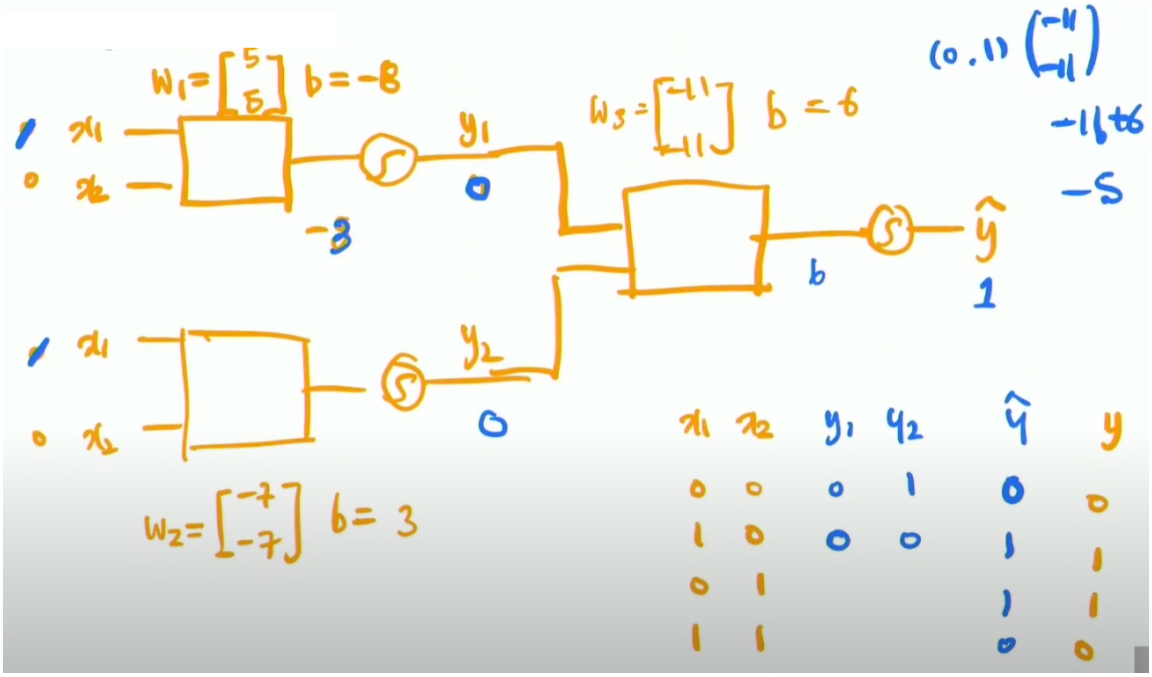


2. Multilayer Perceptron (MLP)

- 용어정의
  - Layers : Input Layer → Hidden Layer → Output Layer
  - Hidden Unit : Hidden Layer에 들어있는 노드
  - 화살표 : Weight를 취하고 + 다음 Layer 직전에 Activation Function을 취한다

▼ XOR Problem Solving

- 다음과 같은 MLP구조를 만들고, XOR Problem이 해결되는지를 살펴보자



① Input 값 $(X_1 = 0, X_2 = 0)$

- 첫번째 노드 연산 : -8은 0에 가까우므로, $Y_1 = 0$, 3은 1에 가까우므로 $Y_2 = 1$
- 두번째 노드 연산 : $\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -11 \\ -11 \end{bmatrix} = -5$
- Predict : Sigmoid에 -5를 입력하면, -5는 0에 가깝기 때문에 0을 리턴
- Prediction 값과 XOR의 실제 수치와 동일 $X_1 = 0, X_2 = 0$일경우$Y = 0$

이렇게 모든 Input 값을 처리해보면, 실제 XOR 결과값과 동일하게 예측하는 것을 발견할 수 있다

- Input이 여러 개일 때 하나의 행렬 연산으로 이를 표현할 수 있으므로 하나의 Layer는 WX + B로 표현이 가능

  Ex) 첫번째 Layer는 $XW + b = \begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} 5 & -7 \\ 5 & -7 \end{bmatrix} + b$로 표현

  → 이는 곧, Linear Transformation한 결과 값과 같다고 볼 수 있다

  → 이 연산 이후에는 Sigmoid로 Activation Function 적용

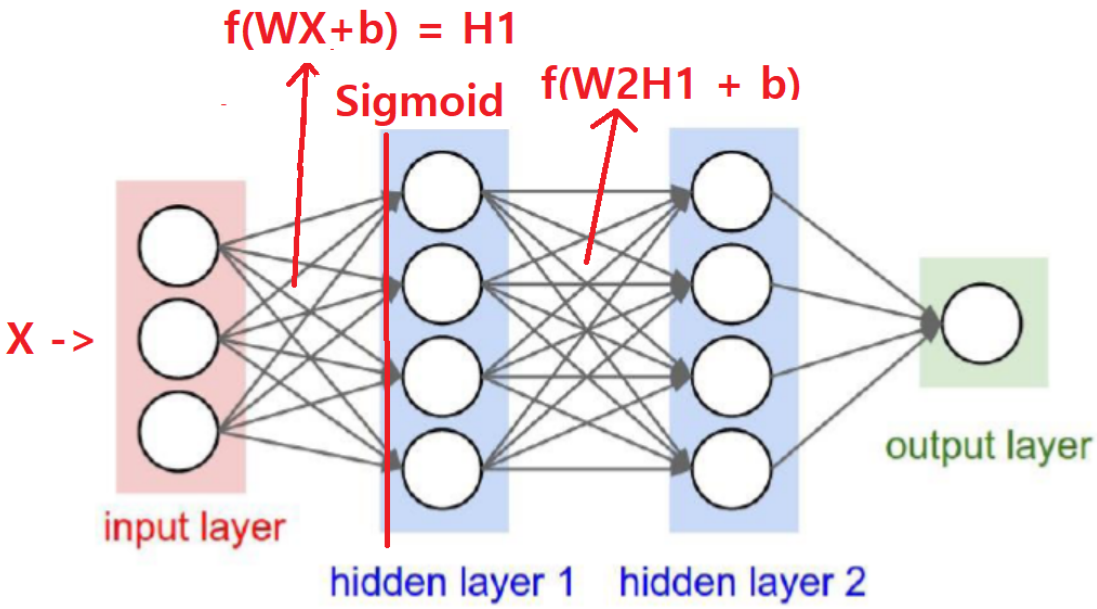
3. Backpropagation

- 배경 : Binary Classification 문제인 XOR가 아닌 더 복잡하고 어려운 문제들의 (Multinomial Classification) 경우에는 어떻게 Training을 할 수 있을까
- Universal Approximation Theorem

  MLP가 엄청나게 Deep해지거나 Layer의 Width가(Hidden Unit) 커지면 어떤 Function이라도 Approximation할 수 있다는 것을 수학적으로 증명됨

- Hypothesis

  다음과 같이 기존의 Hypothesis를 그대로 사용 가능



- Cost Function

  Task에 따라서 MSE, Cross Entropy 어느 것도 적절히 사용가능

- Optimization

  Gradient Descent 알고리즘 사용

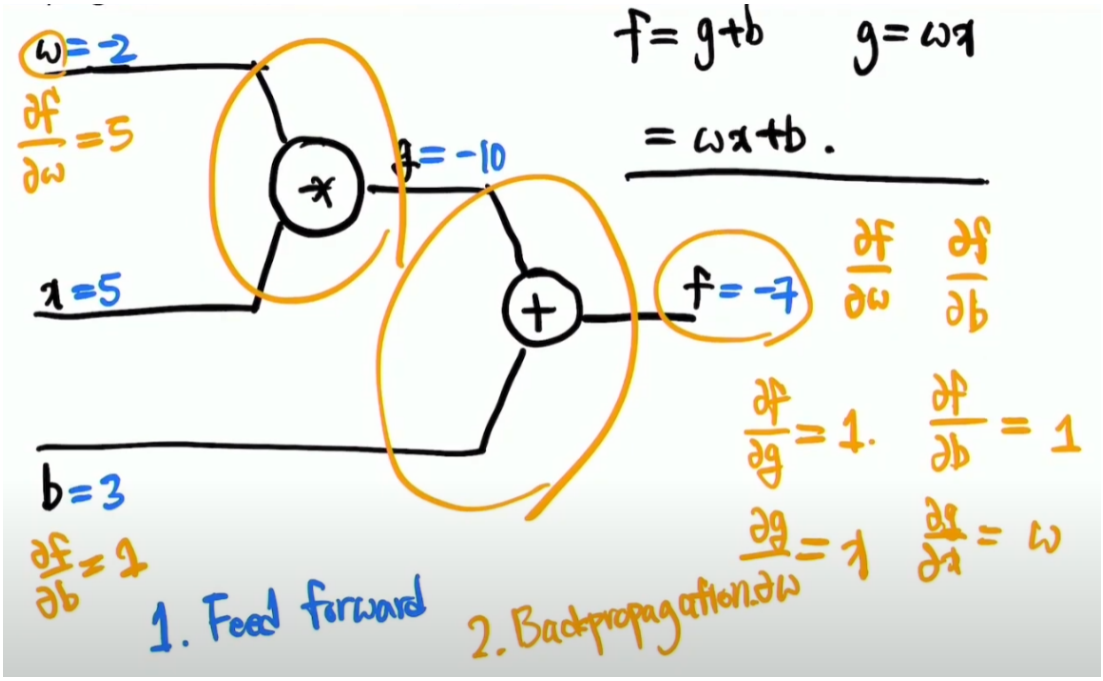
  지난번엔 편미분을 통해 Loss가 W에 얼마나 영향을 미치는가를$\left(=\frac{\partial L}{\partial w}\right)$ 알아보았지만, Layer가 여러개일 경우엔 복잡하다. 하지만 이도 가능하다는 것을 증명할 수 있음

  ▼ 증명) 모델링한 뉴런의 식이 다음과 같다

F = g + b, (g= wx),

F = wx + b

이러한 수식을 Computational Graph로 표현 가능

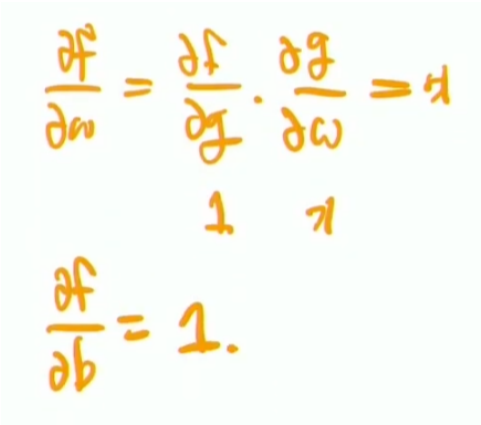


두 가지 스텝으로 뉴런의 Training이 이루어진다

① Feed Forward : Computational Graph의 Input을 활용하여 Output 값을 계산하는 과정

② Backpropagation : Output 값을 각각의 파라메터 값에 대해서 맨 뒤에 있는 Loss가 얼마나 영향을 미치는 알아내는 과정

- Output F를 g로 편미분하고, Output F를 b로 편미분 한다
- g를 w에 대해서 편미분하고, g를 x에 대하여 편미분 한다
- Chain Rule에 의해서 각 노드에서 계산한 편미분 값들을 곱한다

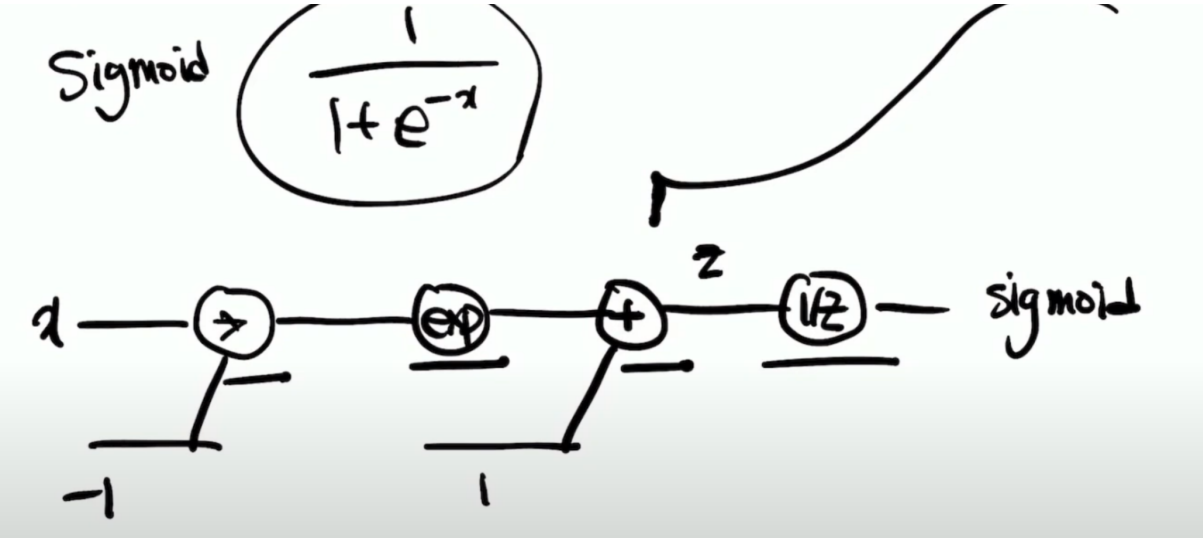


Chain Rule에 의하여 편미분 값들을 곱

③ Gradient값을 알아 낸 후에는 Learning Rate를 구해서 전체 모든 파라메터에 적용하여 Loss를 줄일 수 있다

- Sigmoid 함수도 적용가능

  Input X, -1를 입력받아, Exponential 함수 적용하고 1을 더해준 뒤 역수를 취해주는 Computational Graph 표현



- 이 모든 과정은 파이토치로 구현되어 있음