



딥러닝 홀로서기#13

주제 : Overfitting, Regularization

링크 :

[#13.Lec] Overfitting, Regularization - 딥러닝 홀로서기

발표자료 링크 : <https://github.com/heartcored98/Standalone-DeepLearning/blob/master/Lec3/Lec3-D.pdf>자료 저장소 링크 : <https://github.com/heartcored98/Standalone-DeepLearn...>


 https://youtu.be/_sz3KTyB9Lk



1. Review from Last Lecture

- Categories of ML Problems
 - Method : Supervised Learning vs Unsupervised Learning
 - Predict Value : Discrete vs Continuous

	Supervised Learning	Unsupervised Learning	Reinforcement Learning
Discrete	Classification	Clustering	Discrete Action Space Agent
Continuous	Regression	Dimensionality Reduction	Continuous Action Space Agent



Semi-Supervised Learning

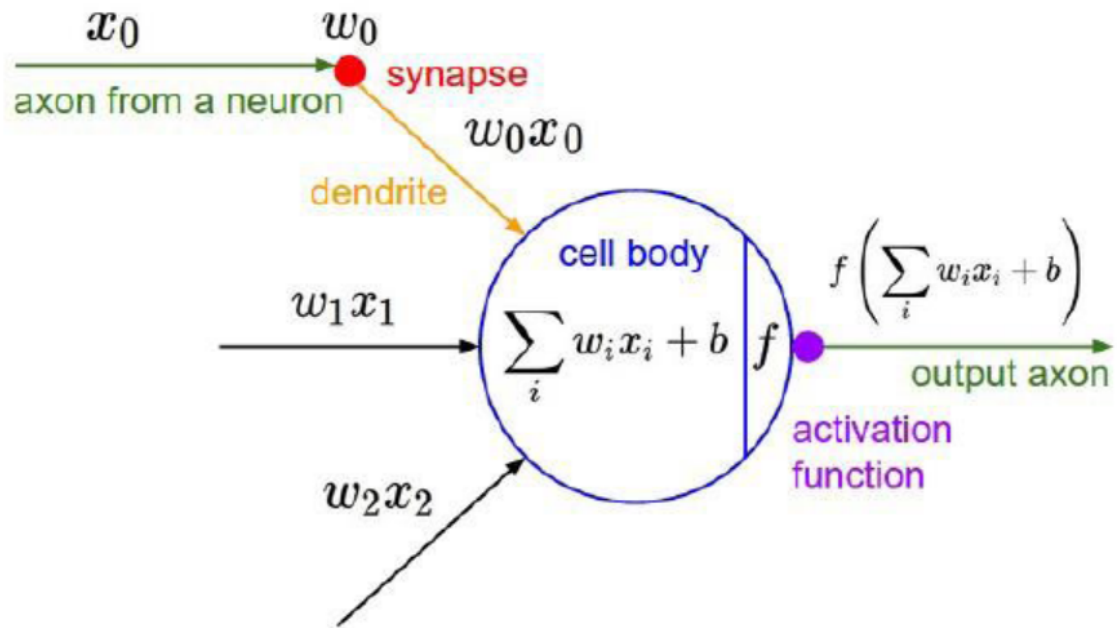
- Binary and Multinomial Classification

- Hypothesis : Linear Regression Function + (WX) Sigmoid Function vs Softmax Function
 - 이런 함수를 씌워서 Continuous한 값을 그 클래스 값이 나올 확률로 변경
- Cost : 결과값은 확률이기 때문에 원래 Linear Regression에서 활용하던 MSE Loss가 아닌, Cross Entropy의 개념을 빌려온 Loss를 활용
- Optimization : Gradient Discent 알고리즘을 사용하여 Training을 수행

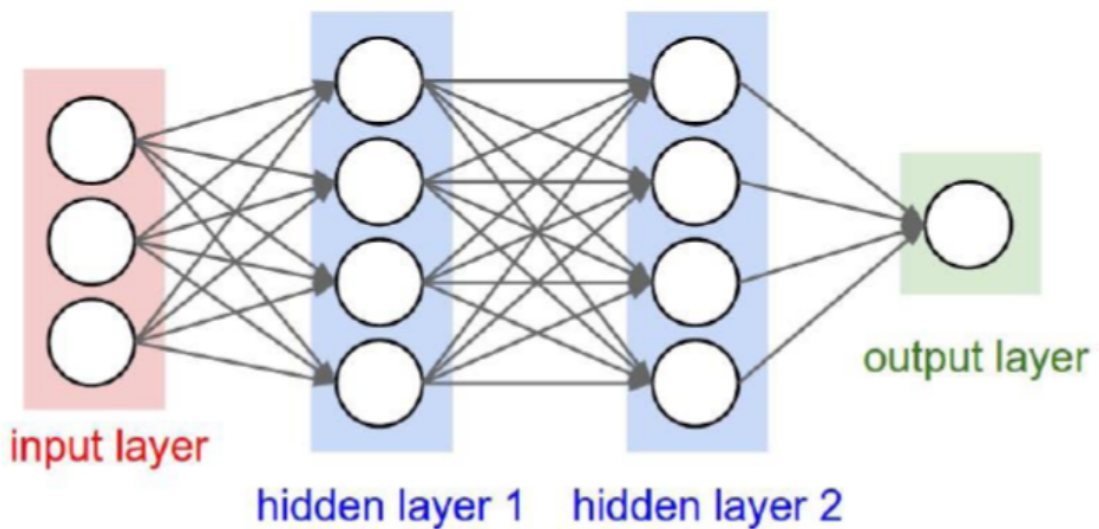
	Binary	Multinomial
Hypothesis	$Sigmoid(WX)$	$Softmax(WX)$

$$\text{Cost} = -y \log(H(x)) - (1 - y) \log(1 - H(x)) \quad \sum -y \log(H(X))$$

- Modeling Neuron
 - 생각하는 기계를 만들기 위해서 뉴런을 수학적 모델링으로 변형
 - 전 뉴런에서 다양한 신호 발생 ($X_1, X_2, X_3 \dots$) → 적당한 Weight 값을 곱해준 뒤 (W_1, W_2, W_3, \dots) → Bias를 더하고 → 적절한 Non-Linear한 Activation Function을 거쳐서 다음 뉴런들의 Input이 되도록 전달



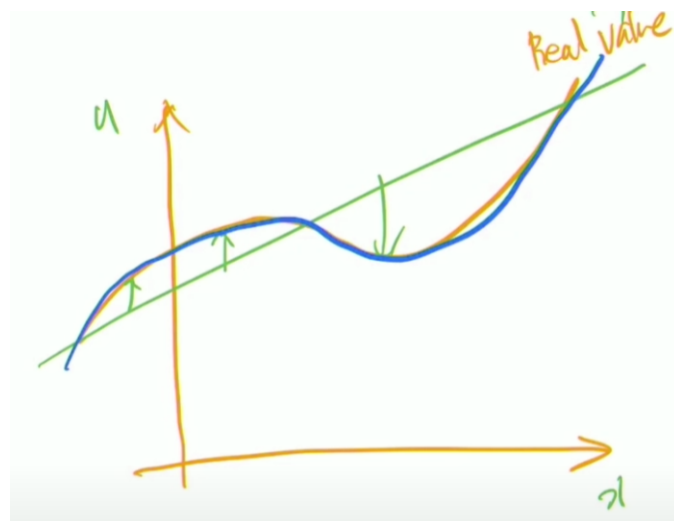
- MLP Structure
 - 위에서 정의한 모델을 여러 Layer로 쌓아놓은 Structure가 MLP Structure이다
 - 이를 이용하여 MNIST Structure를 Training해보면 잘 구동함



2. Problems of MLP - Overfitting

- Deep Learning의 시초와 역사

- MLP라는 개념은 예전에 나왔지만, 민스키 교수가 MLP로 Training 방법이 없다라고 증명
 - 20년동안, AI의 연구가 지지부진하다가, 1980년도에 Backpropagation 알고리즘이 개발됨
 - Backpropagation 알고리즘 이후, MLP가 Training이 가능하다는 것을 발견하였으나, MLP 모델만 가지고는 AI를 제대로 작동시키기 어렵다는 것을 발견
 - 그 문제점 중 하나가 Overfitting 문제임
- Overfitting
 - Model Capacity
 - 예시) Linear Regression을 통해서 어떤 Real Value를 예측해보자
 Linear Regression은($y = wx + b$) 다음과 같은 상황에서 제대로 예측하지를 못한다
 만약, Input값 X를 3차 함수로 표현하면($y = w_1x^3 + w_2x^2 + w_3x + w_4$) Real Value를 더 잘 예측할 수 있을 것이다

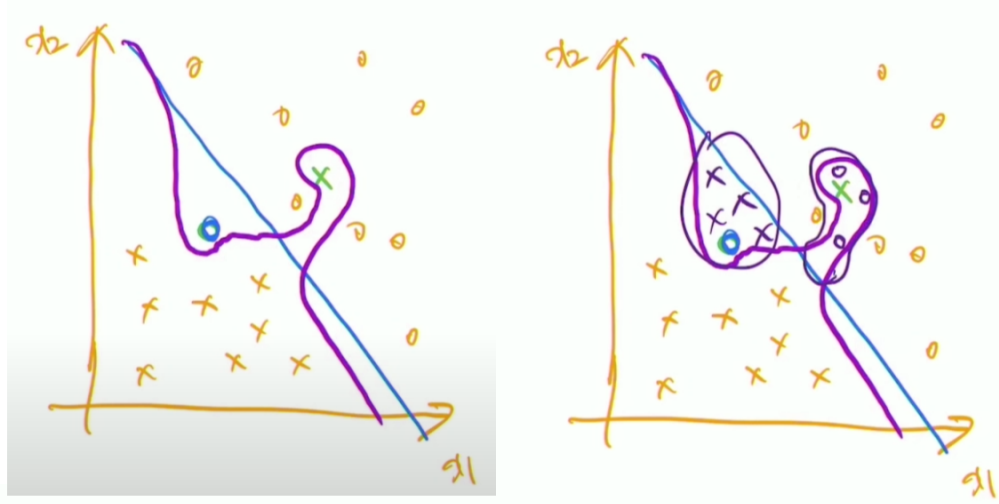


- 이때 두 가지 Hypothesis의 차이점은 파라미터의 갯수이다
 - 앞의 Hypothesis는 파라미터가 2개, 뒤의 Hypothesis는 파라미터가 4개

$$y = w_1x + b \quad 2$$

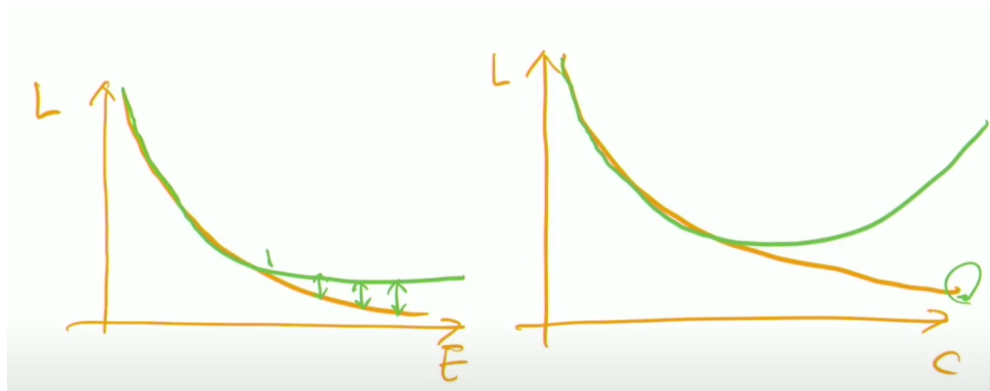
$$y = w_1x^3 + w_2x^2 + w_3x + w_4 \quad 4$$

- 다시말해, 하이퍼파라미터의 갯수를 늘릴 수록(=Model Capacity를 증가) 더욱 더 복잡한 현상들을 잘 예측할 수 있을 것만 같다
- 이를 MLP에 적용해보자
 - 복잡한 현상을 예측하기 위해서 파라미터를 증가 → MLP : Layer를 추가, Hidden Unit 증가
 - 그렇다면 계속하여 무한적으로 모델의 Capacity를 증가시키면 더욱 더 복잡한 문제를 예측이 가능할까?
 - 그렇지 않다라는 것이 결론, 이때 발생하는 문제가 Overfitting 문제!
- 예시2) O와 X로 구성되어 있는 Data Set을 분류하는 문제
 - 심플하게 분류하면 Linear한 선으로 분류하면 좋겠지만, 그럴경우 에러가 발생한다
 - 따라서 모델의 Capacity를 늘려서 분류를 했더니 왼쪽과 같이 에러없이 분류가 가능해졌다
 - 이렇게 학습된 모델을 가지고 Test Data Set을 다시금 분류해 보면, 오른쪽과 같이 다시금 에러가 발생한다



- 다시말해, 학습시킨 Training Data Set만 잘 분류하고 새로 추가된 Test Data Set에는 잘 맞지 않는 모델이 되어버린 것이다
- 이처럼 너무 Fitting을 잘 해버려서 General한 경우에는 적용이 되지 않는 현상을 Overfitting이라 한다
- True Risk and Empirical Risk
 - 어떤 사람 얼굴 사진 속 표정을 분류하는 문제를 가정했을 시,
 - True Risk : 모든 True Distribution(세상에 존재하는 모든 사람들의 표정 데이터)에 대해서 계산한 Error 또는 Loss → 실제로 수행해야하는 문제 → 다만 구할 수 없음
 - Empirical Risk : 일부의 샘플로(몇 백 또는 몇 천장의 얼굴 사진) 계산한 Error 또는 Loss를 통해 True risk를 대표하는 것
 - 즉, True Risk를 줄이기 위해서 Empirical Risk를(Training Set) 활용하여 줄임
 - 위의 예시에서 Overfitting이 일어난 이유는,
 - Empirical Risk가 엄청 작은데에 반해, True Risk가 엄청 커졌기 때문이다
 - Overfitting = Empirical Risk ↓ and True Risk ↑
- Overfitting을 해결하는 방법
 - ① Validation
 - Data Set을 Training Set과 Validation Set로 분류하여 학습 시 Overfitting이 발생하지 않는지 Validation Set을 통해 검증해야함

- 만약, Overfitting이 발생할 경우, Epoch값이 증가할 수록(학습횟수), Training Loss와 Validation Loss의 간극이 발생함 (왼쪽 그림)
- 모델 Capacity를 키우면 키울 수록, Training Loss와 Validation Loss의 간극이 발생함 (오른쪽 그림)
- 따라서, Validation을 통해 적절하게 Training을 끊어주는 것 + 적절한 모델 Capacity를 찾는 것을 수행해야 한다



- Summary
 - Overfitting이란 것을 모델이 Training Dataset을 외워버렸을 경우, General Dataset을(=Test set) 적용하지 못하는 것을 말함
 - 모델 Capacity가 커지면 복잡한 문제를 더욱 정교하게 예측할 수는 있지만, 동시에 Training Set을 외워버려 Overfitting 문제를 야기할 수 있음
 - Empirical Risk를 표현하는 Training Loss와 Approximated True Risk를 표현하는 Validation Loss를 실시간으로 비교하여 Overfitting이 실제로 발생하는 지 알아낼 수 있음

② Regularizations - L2 Regularization and Dropout

- Regularization : 패널티를 준다, 제한한다, Generalize한다
 - 패널티 : 외우는 것을 패널티를 통해 제한
 - Generalize : 특정한 Training Data Set에만 제한하지 않고 General Dataset에도 적용할 수 있도록 함
- Norm

- Norm은 벡터의 길이 또는 크기를 측정하는 방법(함수)
 - Norm이 측정한 벡터의 크기는 원점에서 벡터 좌표까지의 거리 혹은, Magnitude라고 함
- $$L_p = (\sum_i^n |x_i|^p)^{\frac{1}{p}}$$
- P는 Norm의 차수를 의미. (P가 1이면 L1 Norm이고, P가 2이면 L2 Norm)
 - n은 대상 벡터의 요소 수
 - Norm은 각 요소별로 요소 절대값을 p번 곱한 값의 합을 p제곱근한 값

- L2 Regularization
 - Linear Regression의 경우 Loss계산을 위해 MSE를 사용했었으나, 여기에 Regularization 항 $R(\lambda)$ 을 추가

$$Loss = MSE + R(\lambda)$$

이고,

$$R(\lambda) = \lambda ||w||^2$$

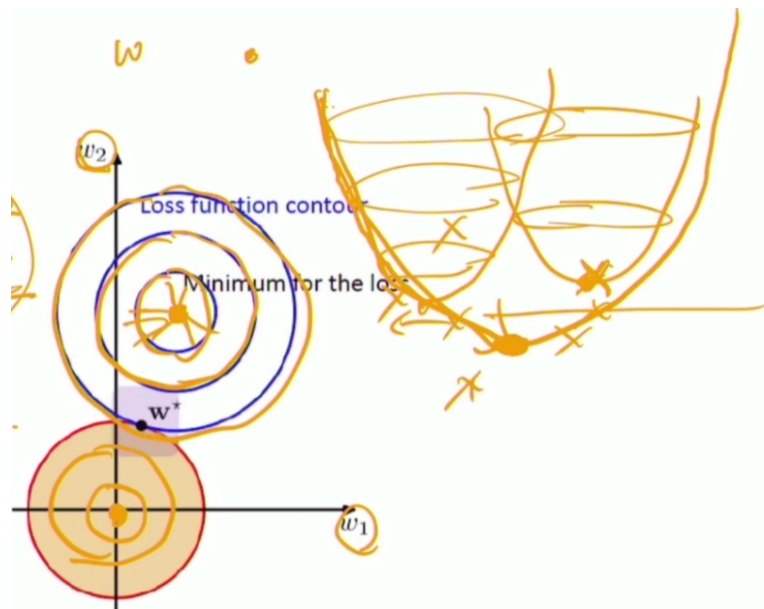
(* λ : 상수 (Constant Value))

- L2 Norm : n차원 좌표평면(유클리드 공간)에서의 벡터의 크기를 계산하여 파라미터의 크기가 너무 발산(커지지) 않도록 제한을 두는 것. 유클리드 Norm이라고도 함

$$L_2 = \sqrt{\sum_i^n x_i^2} = \sqrt{x_1^2 + x_2^2 + x_3^2 + \dots + x_n^2}$$

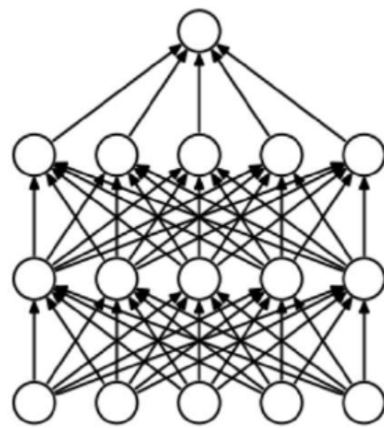
- L2 Norm의 예시 : 피타고라스의 정리
- L2 Norm은 각 요소별로 절대값을 제곱한 수들의 합의 제곱근
- 어떻게 L2 Regularization이 overfitting을 감소시켜 주는가?
 - w_1, w_2 파라미터로 구성된 모델을 어떤 Training Set에 학습을 시킨다고 치자

- 해당 Training Set은 Global Minimum이 있을 것이다
- 만약, Regularization 항이 없다고 가정하면, MSE는 Global Minimum으로 향할 것이다
- 하지만, Global Minimum은 해당 Training Set에만 해당될 것이며, 다른 Dataset을 추가하면, 또 다른 Global Minimum이 새로 생긴다
- 따라서 Regularization 항을 추가하면, 이 Regularization 항은 원점에서 Global Minimum을 가지므로, 실제로 Training시 기존 Training Set의 Global Minimum과 Regularization항의 Global Minimum이 합친 그 사이 어딘가의 적절한 타협점으로 새로운 Global Minimum으로 향하게 된다

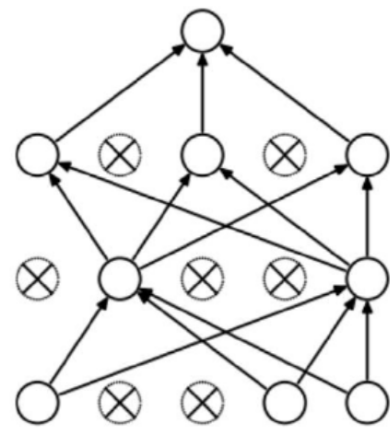


- Dropout
 - 정의 : 임의의 P의 확률로 각각의 노드를 꺼버리는 것 (=노드 상의 weight값 (w)를 0으로 세팅)
 - Training과정 중 Model Capacity를 감소시켜 주는 것 (파라미터의 수를 줄여주는 것)
 - 옆의 노드의 w 가 0으로 세팅되어 있을 경우, 0이 아닌 노드들의 수가 더욱 의미가 있으므로, 각각의 뉴런을 효율 것으로 사용할 수 있게함

- 복잡한 하나의 모델로 예측하기보다, 간단한 여러 모델로 예측하여 그 결과를 합쳐서 사용 (=Ensemble기법)
 - P의 확률로 꺼지는 노드가 Random하게 변화하므로 여러 모델을 생성하게 된다
- 최종적으로 Test set으로 예측할때에는 모든 노드를 켜고 함
- 이로 인하여 Overfitting 문제를 해결



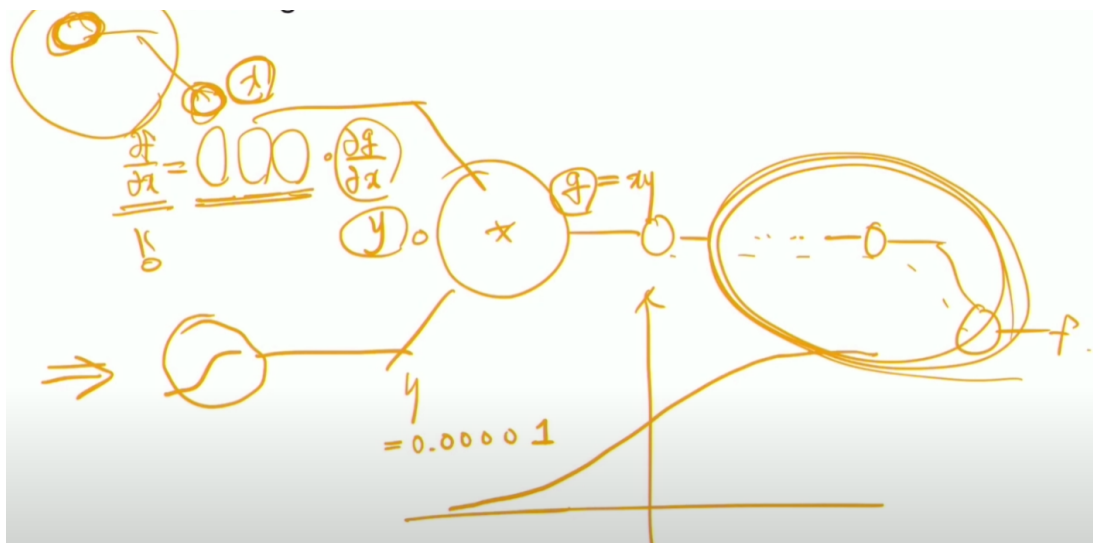
(a) Standard Neural Net



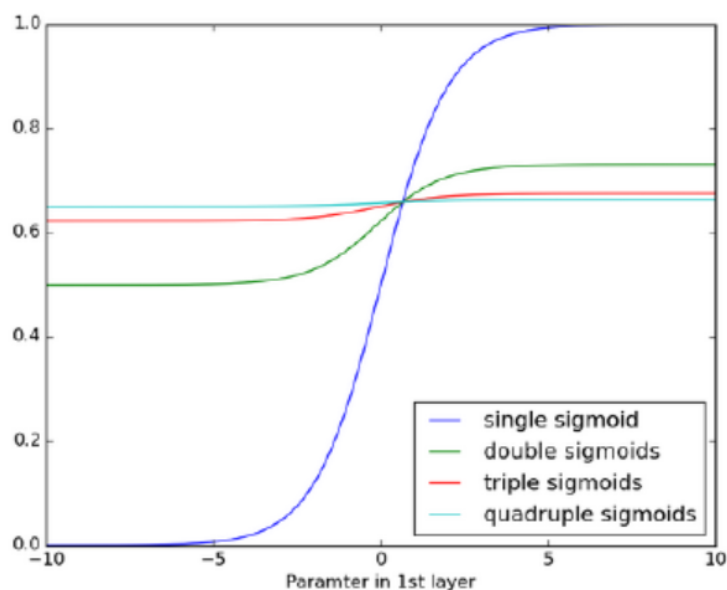
(b) After applying dropout.

3. Problems of MLP - Gradient Vanishing

- Backpropagation 알고리즘에서의 Gradient Vanishing 문제



- Input X를 파라미터라 가정하면, X를 업데이트를 하기 위해서는 Loss에 대한 X의 편미분 값 (=Gradient)를 알아야 업데이트가 가능
- F에서부터 Gradient를 구하면서 거꾸로 이를 계속 곱해주다보면 G에 대한 X의 편미분 값($G=XY$ 이므로)이 Y가 나옴
- 하지만 만약, Y값이 Sigmoid 함수를 거쳐 나왔다고 가정해보면, Sigmoid에 엄청 작은 값이 Input으로 들어갔다고 하면, 여기서 Backpropagation과 정에서 Gradient를 계산하면 앞에 어떤 값이 Input되었더라도 무조건 0으로 수렴해버린다
- 따라서, F에서 가까운 값들은 Training이 되지만, 그 아주 작은 값이 Input으로 입력되는 Sigmoid 다음 Step의 Gradient값들은 무조건 0이 되어버리므로 Gradient가 사라져버려 움직이지 않게된다
- 쉽게 이해하자면, 여러 Layer로 이루어져 있는 MLP의 경우 Sigmoid (Sigmoid (Sigmoid (...))) 의 구조로 Deep해지면 해질 수록, 그 기울기가 점점 0으로 수렴해버리는 현상

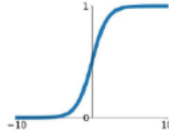


- Gradient Vanishing 문제 해결방법 - Rectified Linear Unit (ReLU) Activation
 - ReLU : 0보다 작으면 0, 0보다 크면 그 값을 그대로 전달
 - Layer가 깊해질 수록, 그 성능이 향상됨
 - 그 밖의 Activation Functions
 - tanh : sigmoid 함수 y값의 범위를 -1 ~ 1까지로 변경한 함수

- Leaky ReLU : 0보다 작을 경우, Gradient를 조금이라도 줄여서 Linear 하게 작은 값을 리턴
- ELU : Leaky ReLU와는 다르게 0보다 작을 경우, Gradient를 Exponential하게 작은 값을 리턴
- Sigmoid, tanh ↔ ReLU와의 성능 차이는 큼
- ReLU ↔ Leaky ReLU, ELU, Maxout간의 성능 차이는 작음

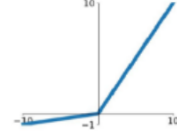
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



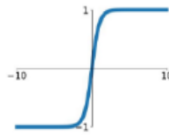
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

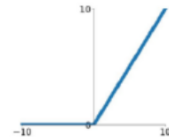


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

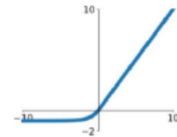
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



4. Other Technic - Xavier Initialization and Batch Normalization

- Xavier Initialization : Weight Initialization의 한 종류, 초기화를 어떻게 하느냐에 따라서 모델의 성능이 달라짐
 - 모든 W를 0으로 초기화 한다면 : Input이 무슨 값이 들어가던지, Loss가 계산이 안됨
 - tanh를 Activation Function으로 사용한다고 가정
 - 초기에 Weight를 Standard Gaussian에서 임의적으로 값을 뽑아 상수 값을 곱해 Initialization
 - 최초에는 Activation Function을 지나친 후, Standard Gaussian모양으로 리턴값들이 형성됨
 - tanh를 계속 지나치면 지나칠 수록, 극단적으로 수렴 → (0으로 수렴) 또는 (1 또는 -1)

- 따라서 Xavier Initialization을 통해서 Weight를 잘 Initialization하면, 이는 계속하여 Gaussian 모양을 유지하는 결과를 얻을 수 있음
- Weight값을 Initialization하는 방식에 따라서도 모델의 성능이 변한다는 것을 알 수 있음
- Batch Normalization
 - MLP 모델에서 하나의 Layer의 Output은 다른 Layer의 Input값으로 전달 되게 됨
 - 이때 Activation Function이 없다면, 각 Layer의 Output W값은 다른 Layer의 Input W'에 다시금 곱해지는 단순한 연산만 하게되고 결국 W값의 변화만 있을 뿐 실질적인 변화는 없어 하나의 Layer를 취한 효과와 동일하다
 - 이와 유사하게, ReLU의 Input값이 너무 편향적으로 위치하여도, 학습이 오래걸리거나 안되는 결과가 발생
 - 따라서, Normalization을 통해 X값들을 정규화하여 적절한 값으로 변경하여 학습에 유의하게 구조를 만들어줌
 - 하지만 언제나 이 방식이 옳은 것은 아니기에, X값들의 분산과 평균값들 또한 학습할 수 있도록 하는 방식을 추가하여 사용

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;	
Parameters to be learned: γ, β	
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$	// scale and shift

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.