M6. 딥러닝 심화

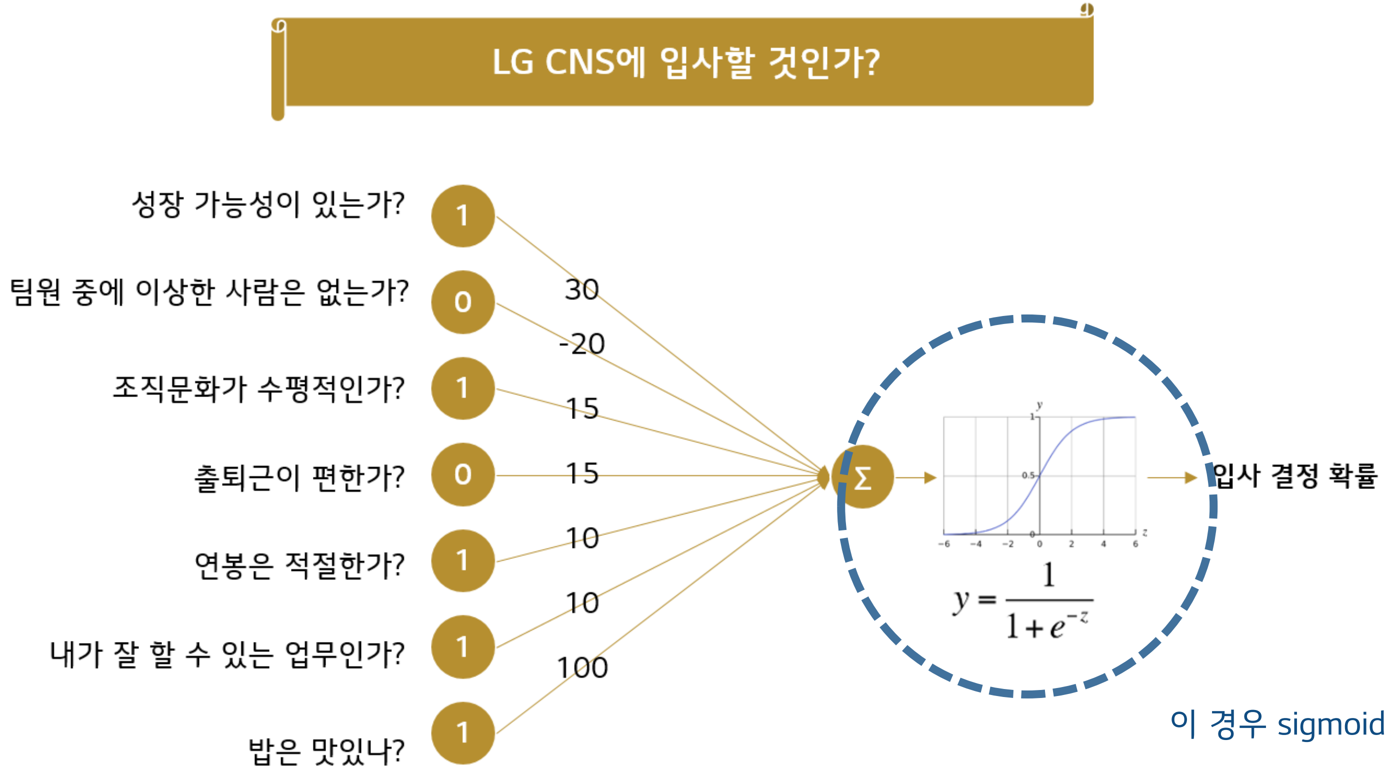
더 나은 심층학습

- Activation Function
- Loss Function
- Optimizer
- Weight Initializer

Activation function

input과 weight의 가중합을 입력 받아 어떤 output을 내보낼지 결정하는 함수

cf. 어제... 이 부분을 기억하십니까..!?



이 경우 sigmoid function이

activation function으로서 채택!

이세상엔 다양한 activation function이 있습니다..

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Logistic (a.k.a. Sigmoid or Soft step)		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$ [1]
TanH		$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$
Inverse square root unit (ISRU)[9]		$f(x) = \frac{x}{\sqrt{1+\alpha x^2}}$
Rectified linear unit (ReLU)[10]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Leaky rectified linear unit (Leaky ReLU)[11]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Parameteric rectified linear unit (PReLU)[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Randomized leaky rectified linear unit (RReLU)[13]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [2]

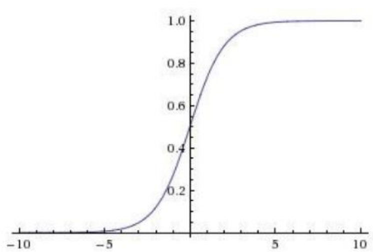
Randomized leaky rectified linear unit (RReLU)[13]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ [2]
Exponential linear unit (ELU)[14]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Scaled exponential linear unit (SELU)[15]		$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$
S-shaped rectified linear activation unit (SReLU)[16]		$f_{t_l,a_l,t_r,a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ t_l, a_l, t_r, a_r are parameters.
Inverse square root linear unit (ISRLU)[9]		$f(x) = \begin{cases} \frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$
Adaptive piecewise linear (APL) [17]		$f(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$
SoftPlus[18]		$f(x) = \ln(1 + e^x)$
Bent identity		$f(x) = \sqrt{}$
Sigmoid-weighted linear unit (SiLU)[19] (a.k.a. Swish[20])		$f(x) = x \cdot$
SoftExponential [21]		$f(\alpha, x) =$

출처 : https://en.wikipedia.org/wiki/Activation_function

자주 이용하는 activation function

Sigmoid

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

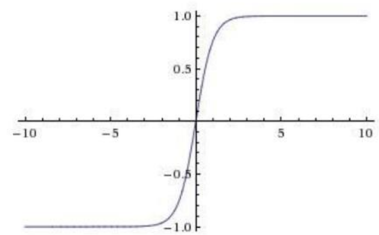


- 입력값을 0~1사이 값으로 변환
- 전통적으로 많이 이용하는 방식

[문제]
- 몹시 크거나 작은 값을 입력으로 받는 경우, gradient가 거의 0에 가까워짐
- 계산이 다소 복잡

Tanh

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



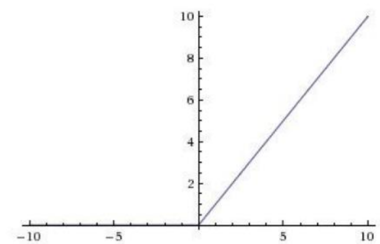
- 입력값을 -1~1 사이 값으로 변환

[문제]
- 역시, 몹시 크거나 작은 값을 입력으로 받는 경우 gradient가 거의 0에 가까워짐

ReLU

*Rectified Linear Unit

$$\mathrm{ReLU}(x) = \max(0, x)$$



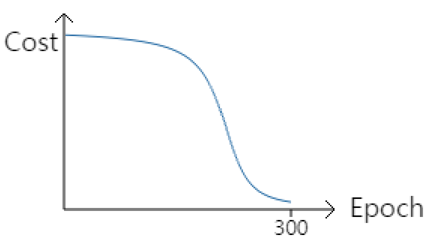
- gradient가 0이 될 일 없음(+구역)
- 매우 간단한 계산
- 실제로 sigmoid나 tanh에 비해 빠르게 수렴하곤 함 (약 6배)

Cost/Loss function

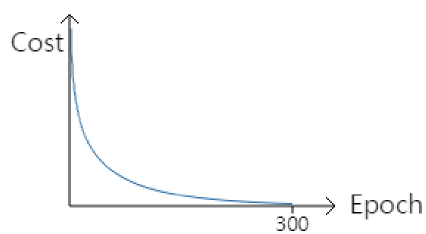
모델의 예측 결과와 실제 정답값의 차이를 정량화해주는 함수

Quadratic(MSE)	Cross Entropy	Negative Log Likelihood
$C(w,b) = \frac{1}{2n}\sum_{i=1}^{n}(a_i - y_i)^2$	$C(w,b) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log a_i + (1-y_i)\log(1-a_i)]$	$C(w,b) = -\sum_{i=1}^{n}\log a_{iy}$
- 전통적인 방식의 loss 계산법 - (-) 하지만 sigmoid와 같이 이용할 경우 수렴을 더디게 함 - (+) 정답이 실수형인 경우에도 사용 가능 (예: regression)	- 이진분류에서 이용되는 함수 - (+) Quadratic Cost에 비해 모델 수렴을 빠르게 함 - 최근 DL framework에서 Multiclass NLL loss와 거의 동일하게 사용	- Multi-class 분류에서 이용되는 함수 - 정답에 해당하는 class만을 고려 - 이진분류의 경우 cross-entropy loss와 동일 - (+) Softmax 함수와 결합하여 사용할 때 모델 수렴을 빠르게 함

동일 모델에 다른 Cost function 적용해볼 때 수렴 속도 차이



Quadratic 적용



Cross entropy 적용

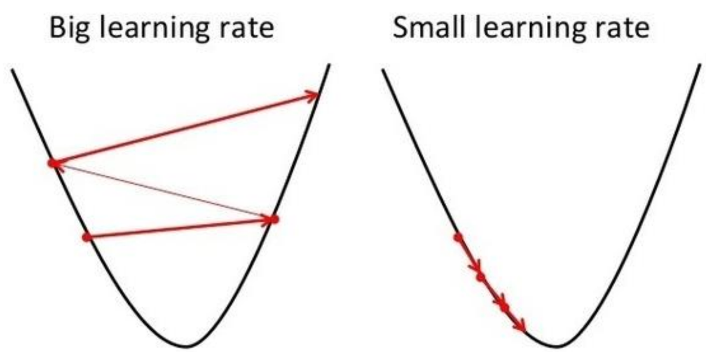
참고 : http://neuralnetworksanddeeplearning.com/chap3.html

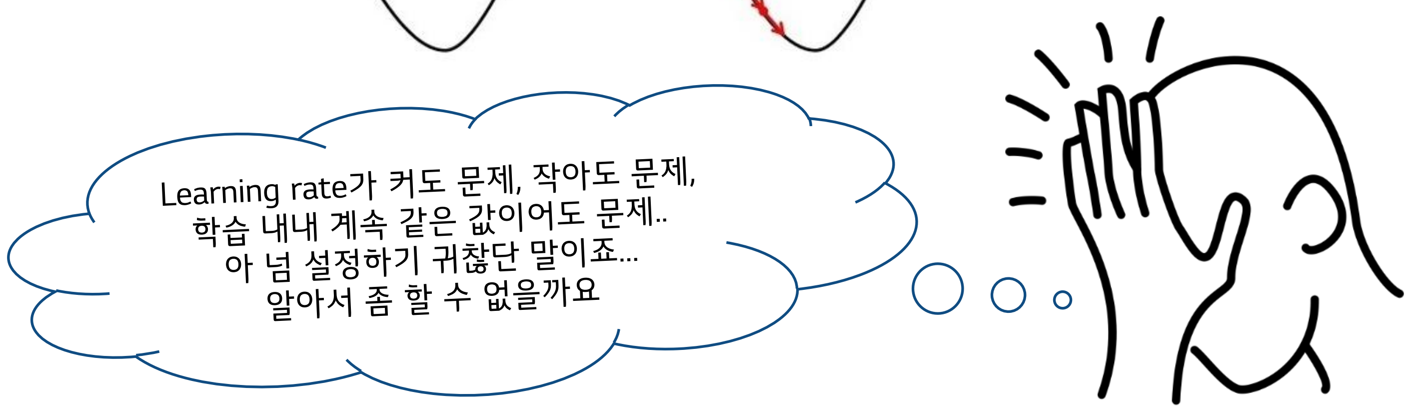
Optimizer

Gradient 정보를 이용해 모델을 더 나은 방향으로 update하는 방법

2단원에서 배운 Gradient descent를 떠올려봅시다

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



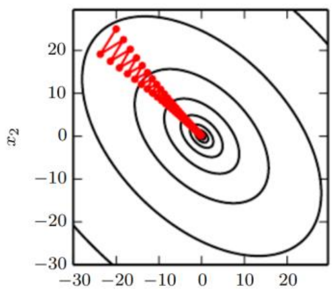


Optimizer

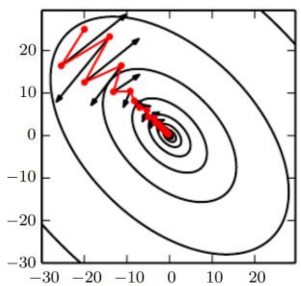
왜 없습니까! 다 방법이 있지요!!

Momentum 계열 optimizer

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



SGD without momentum



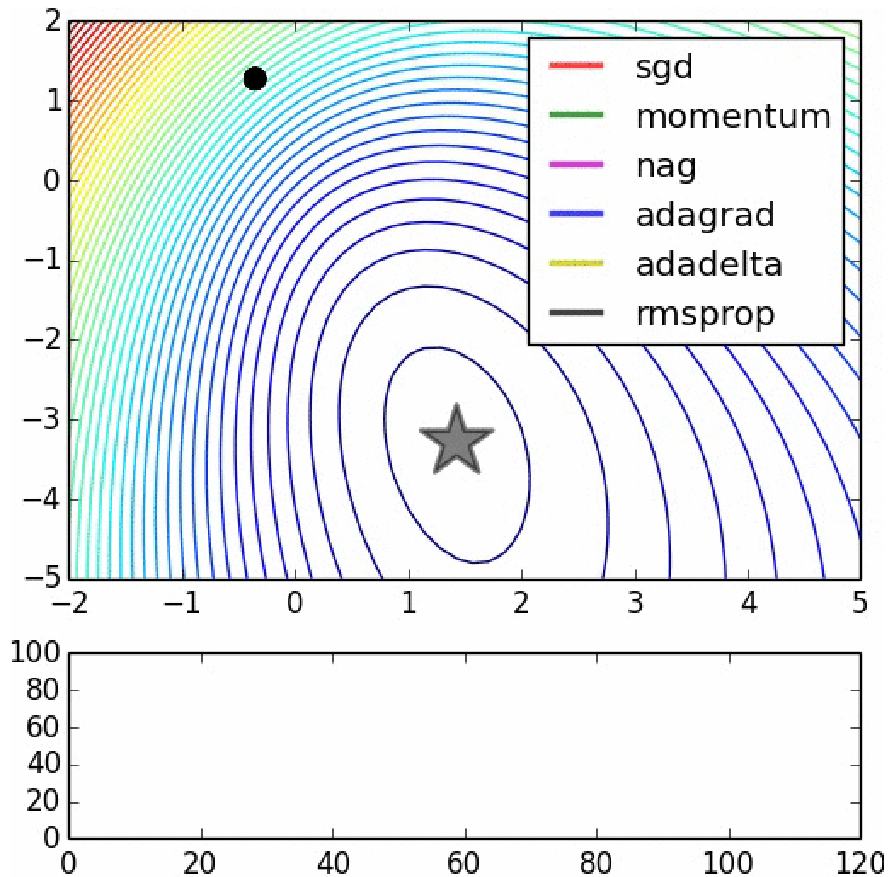
SGD with momentum

Adaptive 계열 optimizer

파라미터 업데이트가 진행될수록 learning rate가 점점 줄어들어 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

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

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

2. Xavier(Glorot) Initialization : 입력/출력 노드 수를 고려하여 초기값을 설정하는 방법, sigmoid/tanh를 쓰는 경우 효과적.

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

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

3. He initialization : Xavier Initialization이 ReLU에서 발생시키는 문제를 해결한 초기화방법, ReLU를 쓰는 경우 효과적.

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

[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!