Deep learning

Sang Yup Lee

신경망에서의 경사하강법

신경망 작동 원리

- 학습
  - 즉, 비용함수를 최소화하는 가중치의 값 찾기
  - Optimization problem
    - 주요 방법
      - Normal equation: appropriate when the cost function is convex
        - But, usually the cost function of DL is not convex (a lot more complex)
        - or too many parameters
      - Newton-Raphson method
        - 경사하강법과 비슷, 하지만 계산 시간이 더 오래 걸려 => 비용함수를 두번 미분해야하기 때문 => 잘 안쓰임
        - https://en.wikipedia.org/wiki/Newton%27s_method
      - Gradient descent
        - 경사하강법은 다음과 같은 식을 통해서 가중치들의 값을 여러번 업데이트 하면서 비용함수의 값을 최소화하는 가중치 값을 찾는 방법 ($w_{1,1}$의 경우)
        - $w_{1,1,new} = w_{1,1,current} - \eta \frac{\partial E}{\partial w_{1,1}}$

신경망 작동 원리

- 경사하강법의 종류
  - 경사하강법은 한번 업데이트 할 때 사용되는 관측치의 수에 따라 크게 세 가지로 구분
    - Batch Gradient Descent
      - 업데이트를 한번 할 때 마다 (비용함수의 값을 계산하기 위해서) 전체 데이터를 모두 사용하는 방법
      - 파라미터의 값이 안정적으로 수렴되지만, 계산하는데 시간이 많이 걸리고 많은 메모리 공간 필요
    - Stochastic Gradient Descent
      - 업데이트를 한번 할때, 랜덤하게 선택된 하나의 관측치만을 사용
      - 계산 속도가 빠르다는 장점이 있으나, 전체 학습 데이터의 특성을 잘 반영하지 못하기 때문에 업데이트 되는 방향이 일정하지 않아, 수렴하는데 시간이 오래 걸린다는 단점 존재

신경망 작동 원리

- 경사하강법의 종류
  - 경사하강법은 한번 업데이트 할 때 사용되는 관측치의 수에 따라 크게 세 가지로 구분
    - Mini-batch Gradient Descent
      - BGD아 SGD 가 갖는 단점을 보완하기 위해서 사용되는 방법
      - 한번 업데이트할 때 하나의 전체 데이터의 일부 (이를 mini-batch 라고 함)를 사용하고, 그 다음 업데이트를 하기 위해서 또 다른 mini-batch를 이용해서 비용함수 (혹은 기울기)의 값을 구하는 방법

신경망 작동 원리

- 경사하강법
  - 비용함수: 설명을 위하 Squared error 비용함수 사용
  - 비용함수의 형태
    - For a single data point
      - $E_i = (y_i - \hat{y}_i)^2$
      - 미분했을 때 발생하는 2를 없애기 위해서 앞에 ½를 곱하기도 함 즉, $E_i = \frac{1}{2}(y_i - \hat{y}_i)^2$
    - For the total data points (in the training dataset, batch data, # of data points = N)
      - $\mathrm{E} = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$
    - For a mini batch (i.e., # of data points = h, where 1<h<N)
      - $\mathrm{E} = \sum_{i=k}^{k+h} E_i = \sum_{i=k}^{k+h} (y_i - \hat{y}_i)^2$

신경망 작동 원리

- 경사하강법 (cont'd)
  - 모형: 선형회귀모형 => $\hat{y}_i = b_1 X_i$
  - 비용함수: Squared errors
  - Toy 학습 데이터 (# of data points = 4)

X	y	$\hat{y}$	E_i
2	5	$2b_1$	$(5-2b_1)^2$
1	2	b_1	$(2-b_1)^2$
3	6	$3b_1$	$(6-3b_1)^2$
4	6	$4b_1$	$(6-4b_1)^2$

  - Total SE
    - $\mathrm{E} = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 = (5-2b_1)^2 + (2-b_1)^2 + (6-3b_1)^2 + (6-4b_1)^2$

신경망 작동 원리

- SGD
  - Random하게 data point (i 번째 데이터 포인트)를 하나 선택. 이를 이용해서, 아래 공식을 통해 파라미터 (가중치)의 값을 한번 업데이트
    - $b_{1,new} = b_{1,current} - \eta \frac{\partial \mathrm{E}_i}{\partial b_1}$
    - E_i=> i번째 데이터 포인트의 비용함수값
  - 예) 첫번째 데이터 포인트가 뽑힌 경우
    - $E_i = E_1 = \frac{1}{2}(5 - 2b_1)^2$
    - $\frac{\partial \mathrm{E}_i}{\partial b_1} = \frac{\partial \mathrm{E}_1}{\partial b_1} = -2(5 - 2b_1) => 4b_1 - 10$
    - 따라서, $b_{1,new} = b_{1,current} - \eta(4b_{1,current} - 10)$
    - 만약 $\eta = 0.1$이고 $b_{1,current} = 10$ 이라면?
      - 10 – 0.1(40-10) = 7 => $b_{1,new}$

learning rate는 hyperparameter이고
가중치 (혹은 파라미터)의
초기값은 랜덤하게 결정됨

신경망 작동 원리

- SGD (cont'd)
  - 그 다음 다시 새로운 데이터 포인트를 random 하게 추출; 만약 두번째 데이터 포인트가 선택되었다면, 아래와 같이 계산
  - $\frac{\partial \mathrm{E}_i}{\partial b_1} = \frac{\partial \mathrm{E}_2}{\partial b_1} = -1(2 - b_1) => b_1 - 2$
  - 따라서 다음과 같이 업데이트
    - $b_{1,new} = b_{1,current} - \eta(b_{1,current} - 2)$
    - 여기에서 $\eta = 0.1$이고 $b_{1,current} = 7$ 이므로
    - $b_{1,new} = 7 - 0.1(7 - 2) = 6.5$
  - 이러한 과정을 반복

여기에서 b1,current는
이전 단계에서 새롭게
업데이트된 값,
즉, 이전 단계에서의 b1,new 가
현재 단계의 b1,current가 됨.

신경망 작동 원리

- BGD
  - 학습 데이터에 있는 모든 데이터 포인트를 사용해서 (아래 식을 이용하여) 파라미터를 업데이트
    - $b_{1,new} = b_{1,current} - \eta \frac{\partial \mathrm{E}}{\partial b_1} = b_{1,current} - \eta \frac{\partial \sum_{i=1}^{N} E_i}{\partial b_1}$
    - 즉, $\frac{\partial \mathrm{E}}{\partial b_1} = 2\{-2(5-2b_1) - 1(2-b_1) - 3(6-3b_1) - 4(6-4b_1)\}$
    - 따라서, $b_{1,new} = b_{1,current} - \eta\{-2(5-2b_{1,current}) - 1(2-b_{1,current}) - 3(6-3b_{1,current}) - 4(6-4b_{1,current})\}$

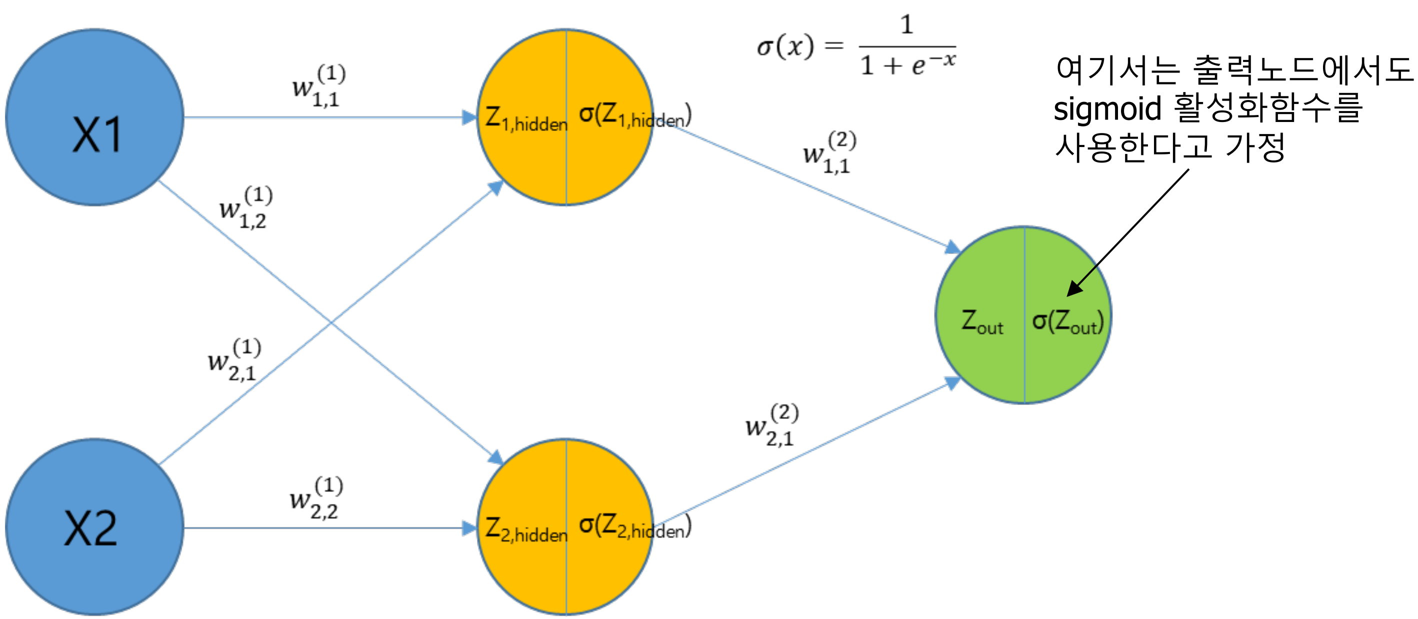
    를 통해서 업데이트 => 이 과정을 반복 => 모든 데이터 포인트에 대해서 업데이트된 파라미터의 값을 가지고 $\frac{\partial \mathrm{E}}{\partial b_1}$을 다시 계산해야 하기 때문에 시간이 올래 걸리는 단점이 있다.

신경망 작동 원리

- Mini batch GD
  - 학습 데이터에 있는 일부 데이터 포인트를 사용해서 (아래 식을 이용하여) 파라미터를 업데이트
    - $b_{1,new} = b_{1,current} - \eta \frac{\partial \sum_{i=k}^{k+h} E_i}{\partial b_1}$
  - 만약 h=2 라고 한다면?
    - 랜덤하게 shuffle 을 한다음에 앞에서부터 2개씩 사용

신경망 작동 원리

- 신경망에서의 경사하강법
  - Example model (설명을 위해 bias node는 생략)



신경망 작동 원리

- 신경망에서의 경사하강법
  - Example model (cont'd)
    - 앞의 모형에서
      - $Z_{1,hidden} = w_{1,1}^{(1)} \cdot X1 + w_{2,1}^{(1)} \cdot X2$
      - $Z_{2,hidden} = w_{1,2}^{(1)} \cdot X1 + w_{2,2}^{(1)} \cdot X2$
      - $h_1 = \sigma(Z_{1,hidden}) = \frac{1}{1+e^{-Z_{1,hidden}}}$
      - $h_2 = \sigma(Z_{2,hidden}) = \frac{1}{1+e^{-Z_{2,hidden}}}$
      - $Z_{out} = w_{1,1}^{(2)} \cdot h_1 + w_{2,1}^{(2)} \cdot h_2$
      - $\hat{y} = \sigma(Z_{out}) = \frac{1}{1+e^{-Z_{out}}}$
    - 각 가중치는 아래 공식을 이용해서 update
      - $w_{1,1new} = w_{1,1\ current} - \eta \frac{\partial E}{\partial w_{1,1}}$

신경망 작동 원리

- 그렇다면 신경망의 경우, $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}}$를 어떻게 계산하는가?
  - 이를 위해서 chain rule을 알아야 한다.
  - 합성함수
    - 아래의 두 개 함수 가정
      - $\mathrm{y} = f(\mathrm{z})$
      - $\mathrm{z} = g(\mathrm{x})$
    - 합성함수: $\mathrm{y} = f(g(\mathrm{x}))$
    - x의 변화는 어떻게 y값에 영향을 미치는가? 위의 합성함수가 의미하는 것은 x값의 변화가 y값에 직접적으로 영향을 주는게 아니라 1차적으로 z의 값을 변화시키고, 변화된 z의 값이 y 값에 영향을 준다는 것을 의미. 아래와 같이 표현.
      - $\Delta\mathrm{x} \rightarrow \Delta\mathrm{z} \rightarrow \Delta\mathrm{y}$
    - Then, how can we calculate $\frac{\partial \mathrm{y}}{\partial x}$?

Δ는 델타라고 하며 변화량을 의미. 즉, x의 변화량이 z를 변화시키고, z의 변화량이 y를 변화시킨다는 것을 의미

x가 1만큼 증가하였을때 y가 얼마만큼 달라지느냐를 의미

신경망 작동 원리

- 합성함수의 미분
  - $\frac{\partial y}{\partial x}$를 구하기 위해서는 연쇄법칙(chain rule)을 사용. 즉,
    - $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x}$
      - $\frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x}$는 x값의 변화가 y값에 영향을 미치는 과정을 보여줌
      - 영향을 주는 방향은 아래와 같음

$$\frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x}$$

$$\longleftarrow$$

신경망 작동 원리

- $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}} \rightarrow w_{1,1}^{(2)}$의 값의 변화 의해서 비용함수의 값이 얼마만큼 달라지는가를 의미
- 하지만, $w_{1,1}^{(2)}$ 가 직접적으로 비용함수의 값에 영향을 주는 것은 아니다.
- 그렇다면 어떠한 과정을 거치는가?

신경망 작동 원리

- 가중치값의 변화는 어떠한 과정을 거쳐서 비용함수에 영향을 주는가?
- 예) $\Delta w_{1,1}^{(2)}$의 경우
  - $\Delta w_{1,1}^{(2)} \rightarrow \Delta E_i$ => 중간에서 어떠한 일이 발생하는가?
  - 아래와 같은 과정을 거침
  - $\Delta w_{1,1}^{(2)} \rightarrow \Delta Z_{out} \rightarrow \Delta \hat{y}_i \rightarrow \Delta E_i$
    - 가중치, $w_{1,1}^{(2)}$, 값의 변화는 Z_{out}에 영향을 주고 Z_{out}의 변화는 $\hat{y}$에 영향을 주고 $\hat{y}$의 변화는 E_i에 영향을 준다.
  - 이를 chain rule로 표현하면,
    - $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}} = \frac{\partial Z_{out}}{\partial w_{1,1}^{(2)}} \frac{\partial \hat{y}_i}{\partial Z_{out}} \frac{\partial E_i}{\partial \hat{y}_i}$
  - 합성함수의 표현
    - $E_i = f(\hat{y}_i)$, $\hat{y}_i = g(Z_{out})$, $Z_{out} = h(w_{1,1}^{(2)})$
    - 따라서 $E_i = f(g(h\left(w_{1,1}^{(2)}\right)))$

신경망 작동 원리

- 가중치 업데이트
  - SGD의 경우 (다른 방법들도 유사)
    - $w_{1,1,new}^{(2)} = w_{1,1,current}^{(2)} - \eta \frac{\partial \mathrm{E}_i}{\partial w_{1,1}^{(2)}}$
    - 그렇다면 $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}} = \frac{\partial Z_{out}}{\partial w_{1,1}^{(2)}} \frac{\partial \hat{y}_i}{\partial Z_{out}} \frac{\partial E_i}{\partial \hat{y}_i}$ 은 어떻게 표현되는가?
    - 즉,
      - ① $\frac{\partial E_i}{\partial \hat{y}_i}$, $E_i = \frac{1}{2}(y_i - \hat{y}_i)^2$

비용함수는 SE라고 가정

계산의 용이성을 위해 ½ 사용

        - 따라서, $\frac{\partial E_i}{\partial \hat{y}_i} = -(y_i - \hat{y}_i) \rightarrow \hat{y}_i - y_i$

신경망 작동 원리

- 가중치 업데이트 (cont'd)
  - ② $\frac{\partial \hat{y}_i}{\partial Z_{out}}$, $\hat{y} = \sigma(Z_{out}) = \frac{1}{1+e^{-Z_{out}}}$
    - How to differentiate?
      - 다음 슬라이드의 미분 공식 참고
    - Then, we get
      - $\frac{\partial \hat{y}_i}{\partial Z_{out}} = \frac{e^{-Z_{out}}}{\left(1+e^{-Z_{out}}\right)^2} = \frac{1}{1+e^{-Z_{out}}}\left(1 - \frac{1}{1+e^{-Z_{out}}}\right) = \hat{y}_i(1 - \hat{y}_i)$
      -

부록: 미분 공식

- 분수 함수의 미분
  - $\mathrm{y} = \frac{f(x)}{g(x)}$
  - $\mathrm{y}' = \frac{f'(x)g(x) - f(x)g\prime(x)}{g(x)^2}$
- 지수함수 미분
  - $\mathrm{y} = \mathrm{a}^{f(x)}$
  - $\mathrm{y}' = f'(x)a^{f(x)}\ln(a)$
  - 따라서, $y = e^{f(x)}$ => $y' = f'(x)e^{f(x)}$
- 로그함수 미분
  - $\mathrm{y} = \log_a f(x)$
  - $\mathrm{y}' = \frac{f'(x)}{f(x)} \frac{1}{\log_{\mathrm{e}} a} = \frac{f'(x)}{f(x)} \frac{1}{\ln(a)}$

신경망 작동 원리

- 가중치 업데이트 (cont'd)
  - ③ $\frac{\partial Z_{out}}{\partial w_{1,1}^{(2)}}$, $Z_{out} = w_{1,1}^{(2)} \cdot h_1 + w_{2,1}^{(2)} \cdot h_2$
    - 따라서, $\frac{\partial Z_{out}}{\partial w_{1,1}^{(2)}} = h_1$
  - ①,②,③ 을 이용해서
    - $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}} = \frac{\partial Z_{out}}{\partial w_{1,1}^{(2)}} \frac{\partial \hat{y}_i}{\partial Z_{out}} \frac{\partial E_i}{\partial \hat{y}_i} = h_1 \cdot \hat{y}_i(1 - \hat{y}_i) \cdot (\hat{y}_i - y_i)$
    - 이를 이용해서 다음을 계산
    - $w_{1,1,new}^{(2)} = w_{1,1,current}^{(2)} - \eta \frac{\partial \mathrm{E}_i}{\partial w_{1,1}^{(2)}}$

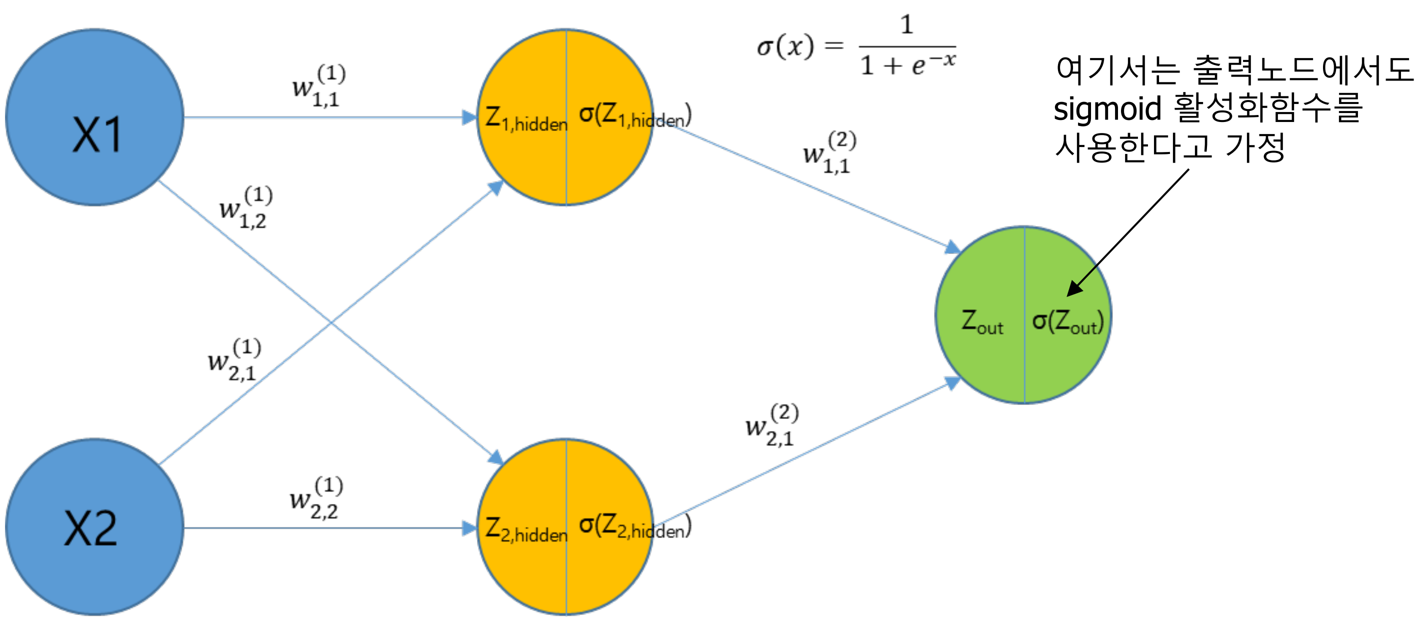
신경망 작동 원리

이부분이 중복 ⇒
따라서 **<u>실제 계산을 할 때는</u>**
이부분 먼저 계산한다.
⇒ 이를 오차역전파라고함

- 또 다른 가중치의 업데이트
- 예) $w_{1,1}^{(1)}$
  - $$\frac{\partial \mathrm{E}_i}{\partial w_{1,1}^{(1)}} = \frac{\partial \mathrm{Z}_{1,hidden}}{\partial w_{1,1}^{(1)}} \frac{\partial h_1}{\partial \mathrm{Z}_{1,hidden}} \frac{\partial \mathrm{Z}_{out}}{\partial h_1} \boxed{\frac{\partial \hat{y}_i}{\partial Z_{out}} \frac{\partial E_i}{\partial \hat{y}_i}}$$
    - $\frac{\partial E_i}{\partial \hat{y}_i} = -(y_i - \hat{y}_i) \rightarrow \hat{y}_i - y_i$
    - $\frac{\partial \hat{y}_i}{\partial Z_{out}} = \hat{y}_i(1 - \hat{y}_i)$
    - $\frac{\partial \mathrm{Z}_{out}}{\partial h_1} = w_{1,1}^{(2)}$
    - $\frac{\partial h_1}{\partial \mathrm{Z}_{1,hidden}} = h_1(1 - h_1)$
    - $\frac{\partial \mathrm{Z}_{1,hidden}}{\partial w_{1,1}^{(1)}} = X1_i$

예제 데이터를 사용한 가중치 업데이트

- 사용 모형 (설명을 위해 bias node는 생략)



예제 데이터를 사용한 가중치 업데이트

- Toy 학습 데이터

X1	X2	y
1	1	1
2	2	0

- 경사하강법: SGD
- 가중치 벡터: 처음에는 랜덤하게 assign
  - $$\mathbf{w} = \begin{bmatrix} w_{1,1}^{(1)} \\ w_{1,2}^{(1)} \\ w_{2,1}^{(1)} \\ w_{2,2}^{(1)} \\ w_{1,1}^{(2)} \\ w_{2,1}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.3 \\ 0.4 \\ 0.2 \\ 0.1 \\ 0.1 \\ 0.2 \end{bmatrix}$$

예제 데이터를 사용한 가중치 업데이트

- 순서
  - (랜덤하게 선택된) 각 관측치에 대해서
    - 단계1: 순전파 (feedforward propagation)
      - 독립변수 정보를 이용해서 비용함수를 계산
      - 즉, $E_i = \frac{1}{2}(y_i - \hat{y}_i)^2$를 계산
      - 처음에는 초기값으로 설정된 파라미터의 값들을 사용
    - 단계2: 역전파 (backward propagation)
      - 앞에서 구한 경사하강법 식 (예, $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}} = h_1 \cdot \hat{y}_i(1 - \hat{y}_i) \cdot (\hat{y}_i - y_i)$)을 사용해서 각 파라미터를 업데이트, 이때 업데이트되는 순서는 에러항 쪽으로부터 역으로

예제 데이터를 사용한 가중치 업데이트

- 순전파 (Forward propagation)
  - 즉, 입력데이터가 입력되어 비용함수가 계산된다.
  - For the first data point,
    - $Z_{1,hidden} = \ 0.3 \cdot 1 + 0.2 \cdot 1 = 0.5$
    - $Z_{2,hidden} = \ 0.4 \cdot 1 + 0.1 \cdot 1 = 0.5$
    - $h_1 = \sigma(Z_{1,hidden}) = \frac{1}{1+e^{-Z_{1,hidden}}} = \frac{1}{1+e^{-0.5}} = 0.6224593312018546$
    - $h_2 = \sigma(Z_{2,hidden}) = \frac{1}{1+e^{-Z_{2,hidden}}} = \frac{1}{1+e^{-0.5}} = 0.6224593312018546$
    - $Z_{out} = w_{1,1}^{(2)} \cdot h_1 + \ w_{2,1}^{(2)} \cdot h_1 = 0.1 \cdot 0.622 + 0.2 \cdot 0.622 = 0.187$
    - $\hat{y}_1 = \ \sigma(Z_{out}) = \frac{1}{1+e^{-0.187}} \approx 0.547$
    - $E_1 = \frac{1}{2}(y_1 - \hat{y}_1)^2 = \frac{1}{2}(1 - 0.547)^2 \approx 0.103$

예제 데이터를 사용한 가중치 업데이트

- 역전파
  - 파라미터의 값을 업데이트
  - 역전파: 즉, $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}}$ 등을 먼저 계산하고 그 다음 $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(1)}}$ 등을 계산
  - $w_{1,1}^{(2)}$에 대해서
    - $w_{1,1,new}^{(2)} = w_{1,1,current}^{(2)} - \eta \frac{\partial \mathrm{E}_1}{\partial w_{1,1}^{(2)}}$
    - $\frac{\partial \mathrm{E}}{\partial w_{1,1}^{(2)}} = \frac{\partial Z_{out}}{\partial w_{1,1}^{(2)}} \frac{\partial \hat{y}_1}{\partial Z_{out}} \frac{\partial E_1}{\partial \hat{y}_1} = h_1 \cdot \hat{y}_1(1 - \hat{y}_1) \cdot (\hat{y}_1 - y_1) = 0.622 \cdot 0.547(1 - 0.547) \cdot (0.547 - 1) = -0.0698$
    - When η (learning rate) = 0.01,
      - $w_{1,1,new}^{(2)} = w_{1,1,current}^{(2)} - \eta \frac{\partial \mathrm{E}_1}{\partial w_{1,1}^{(2)}} = 0.1 - 0.01 \cdot -0.0698 = 0.10069$

예제 데이터를 사용한 가중치 업데이트

- 역전파 (cont'd)

이부분은 이미 계산되어 있음

  - $w_{1,1}^{(1)}$에 대해서
    - $w_{1,1,new}^{(1)} = w_{1,1,current}^{(1)} - \eta \frac{\partial \mathrm{E}_1}{\partial w_{1,1}^{(1)}}$
    - $\frac{\partial \mathrm{E}_i}{\partial w_{1,1}^{(1)}} = X1_1 \cdot h_1(1-h_1) \cdot w_{1,1}^{(2)} \boxed{\cdot\, \hat{y}_1(1-\hat{y}_1) \cdot (\hat{y}_1 - y_1)}$
    - $X1_1 = 1$
    - $h_1 = \sigma\left(Z_{1,hidden}\right) = \frac{1}{1+e^{-Z_{1,hidden}}} = \frac{1}{1+e^{-0.5}} = 0.6224593312018546$
    - $w_{1,1,current}^{(2)} = 0.1$
    - $\hat{y}_1(1-\hat{y}_1) \cdot (\hat{y}_1 - y_1) = 0.547(1-0.547) \cdot (0.547-1)$
    - $\frac{\partial \mathrm{E}_i}{\partial w_{1,1}^{(1)}} = X1_1 \cdot h_1(1-h_1) \cdot w_{1,1}^{(2)} \cdot \hat{y}_1(1-\hat{y}_1) \cdot (\hat{y}_1 - y_1)$

    $= 1 \cdot 0.622 \cdot (1-0.622) \cdot 0.1 \cdot 0.547 \cdot (1-0.547) \cdot (0.547-1) = -0.00264$
    - $w_{1,1,new}^{(1)} = w_{1,1,current}^{(1)} - \eta \frac{\partial \mathrm{E}_1}{\partial w_{1,1}^{(1)}} = 0.3 - 0.01 \cdot -0.00264 = 0.30026$
- We can do the same thing with other parameters.

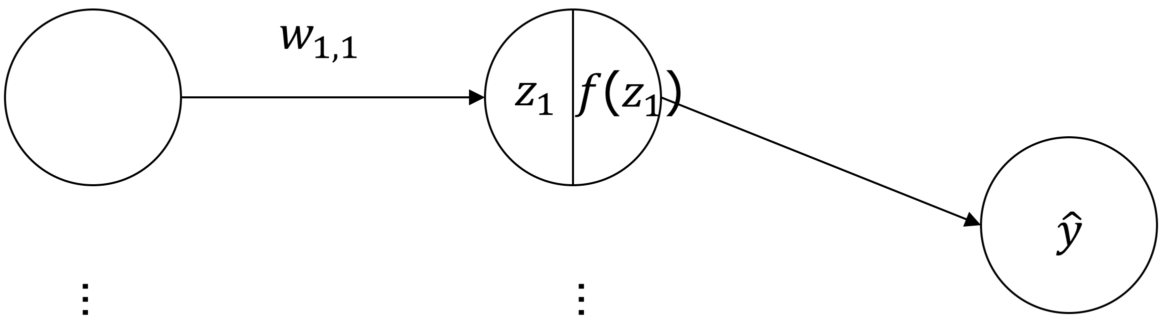
경사 소실 문제

신경망 작동 원리

- 경사하강법
  - 경사 소실 문제 (Vanishing gradient problem) (& 폭발 문제 (exploding problem))
    - 여기에서 "사라진다 (vanish)" 의 의미는 $\frac{\partial \mathrm{E}}{\partial w_{j,i}}$의 값이 0에 가까워진다는 것
    - $w_{j,i,new} = w_{j,i,current} - \eta \frac{\partial \mathrm{E}}{\partial w_{j,i}}$에서 보듯이 $\frac{\partial \mathrm{E}}{\partial w_{j,i}} \approx 0$ 이 되면 가중치의 값이 업데이트 되지 않는 문제가 발생 => 최적의 가중치 값을 찾기가 어려움

신경망 작동 원리

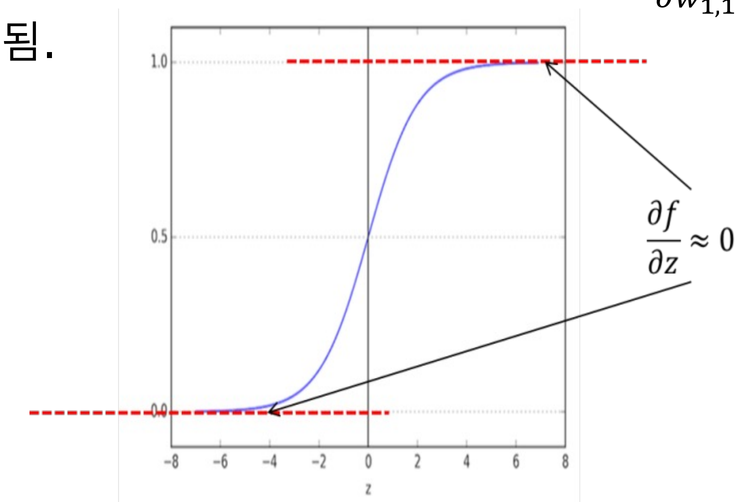
- 경사 소실 문제



- $$w_{1,1,new} = w_{1,1,current} - \eta \frac{\partial \mathrm{E}}{\partial w_{1,1}}$$
  - $$\frac{\partial \mathrm{E}}{\partial w_{1,1}} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f} \cdot \frac{\partial f}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_{1,1}}$$

신경망 작동 원리

- 경사 소실 문제
  - $\frac{\partial \mathrm{E}}{\partial w_{1,1}} = \frac{\partial E}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial f} \cdot \frac{\partial f}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_{1,1}}$
    - 로지스틱 또는 tanh 함수를 활성화 함수로 사용하는 경우에는 $\frac{\partial f}{\partial z_1}$의 값이 0에 가까워질 수가 있는 것. 그렇게 되면 $\frac{\partial \mathrm{E}}{\partial w_{1,1}}$의 값이 0에 가까워지게 됨.



    - 그 대신 Relu나 Leaky relu 함수를 사용

신경망 작동 원리

- 경사 소실 문제
  - 참고
    - ReLU 함수도 $z < 0$인 경우에는 미분값이 0이 되기는 하지만, 관련된 연구에 따르면 $z < 0$인 부분에서 ReLU 함수의 미분값이 0이 되더라도 많은 경우 학습 결과에 큰 영향을 미치지 않으며 오히려 학습 속도를 개선시키는 효과가 있다고 함 (참고: ReLU 함수의 미분값이 0이 되는 현상을 'Dead ReLU'라고 함)
    - 이러한 문제를 보완하기 위해 제안된 활성화 함수
      ⇒ Leaky ReLU와 ELU 등
      - 하지만, ELU의 경우 ReLU에 비해 속도가 느리다는 단점 존재

신경망 작동 원리

- 경사 폭발 문제
  - 경사 소실 문제 보다는 덜 빈번하게 발생
  - 경사 값이 너무 커져서 발생하는 문제
  - 영향
    - 업데이트의 값이 너무 크기 때문에 파라미터의 값이 수렴하지 않고 발산하거나 파라미터의 값이 너무 커지는 문제 발생
    - 비용함수의 값이 업데이트 마다 크게 달라지거나, NaN이 됨
  - 주된 원인
    - 파라미터의 초기값이 커서
  - 해결 방안
    - L1 또는 L2 규제
    - 경사 클리핑 (clipping) 등

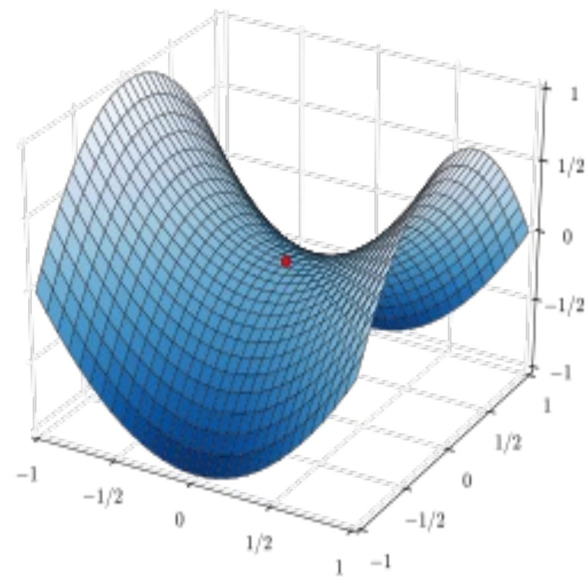
OPTIMIZER의 종류

신경망 작동 원리

- 기본 경사 하강법 (SGD)의 제한점
  - 기본 업데이트 공식 => $w_{j,i,new} = w_{j,i,current} - \eta \frac{\partial \mathrm{E}}{\partial w_{j,i}}$
  - 제한점
    - 지금까지 업데이트된 정도가 반영되지 않는다.
    - 업데이트 횟수와 상관없이 learning rate가 고정되어 있다.

신경망 작동 원리

- 기본적 방법의 문제
  - 1) 비용함수의 saddle point를 잘 벗어나지 못한다.
    - 딥러닝에서는 파라미터가 많아서 local optima는 많이 존재하지 않는다. 대신 오른쪽 그림과 같은 saddle point가 많이 존재한다. 즉, saddle point를 어떻게 벗어나느냐가 중요한 문제이다.
  - 2) 속도가 느리다.



신경망 작동 원리

- Optimizer의 종류와 특성
  - 이러한 문제를 보완하기 위해서 다양한 형태의 optimizer 제안
    - 경사하강법을 이용하여 비용함수의 값을 최소화하는 가중치의 값을 찾는 역할을 하는 것을 optimizer라고 함
  - 기본 업데이트 공식을 약간씩 수정/보완한 방법들임
  - 대표적 Optimizers
    - Momentum
    - NAG (Nesterov Accelerated Gradient)
    - Adagrad (Adaptive Gradient)
    - RMSprop (Root Mean Square Propagation)
    - Adadelta
    - Adam
    - 다른 optimizer들은 https://ruder.io/optimizing-gradient-descent/를 참고

Optimizer

- Momentum
  - 이전 update 정보를 기억, 현재 update에 반영하는 방법

$$v_t = \gamma v_{t-1} + \eta \frac{\partial \mathrm{E}}{\partial w_i}(w_{i,current})$$

$$w_{i,new} = w_{i,current} - v_t$$

- 이전에 업데이트된 정도
- γ는 하이퍼파라미터로 보통 0.9

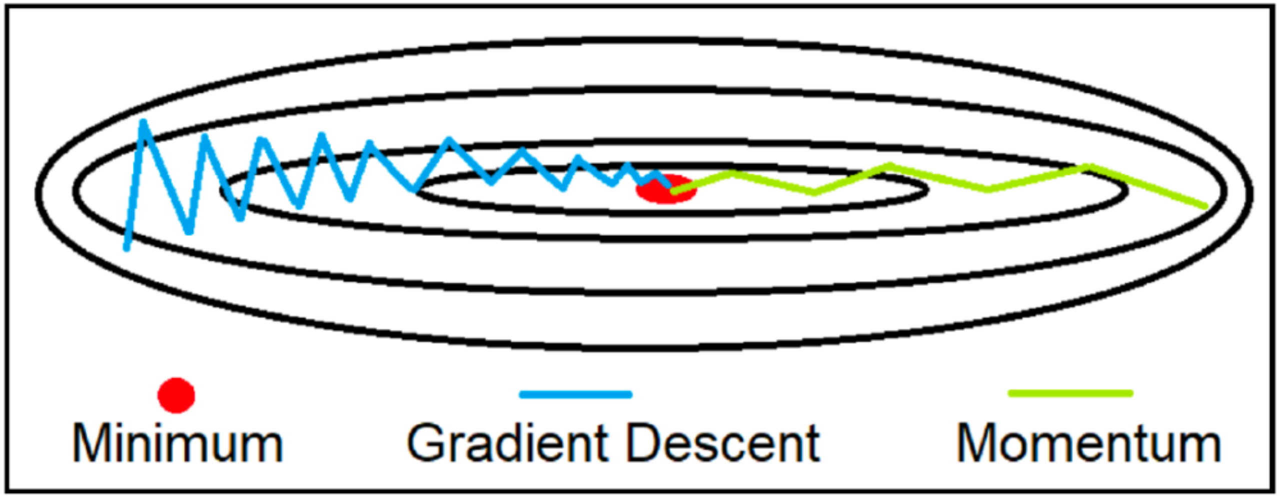
$$w_{i,new} = w_{i,current} - \left(\boxed{\gamma v_{t-1}} + \eta \frac{\partial \mathrm{E}}{\partial w_i}(w_{i,current})\right)$$

  - 장점 (SGD 에 비해서)
    - 속도가 빠르다.
  - 하지만, 여전히 안장점을 잘 벗어나지 못하는 문제 존재

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, *12*(1), 145-151.

Optimizer

- Momentum (cont'd)
  - 효과



Optimizer

- NAG (Nesterov Accelerated Gradient, Nesterov momentum이라고도 표현)
  - 모멘텀 방법은 업데이트가 조금만 발생해야 하는 지점에서 상대적으로 많이 발생한다는 단점 존재
  - 현재의 파라미터 값에서의 경사를 계산해서 업데이트에 사용하는 것이 아니라, 파라미터가 새롭게 업데이트될 지점에서의 경사를 사용해서 현재 단계의 업데이트를 진행하는 방법

$$v_t = \gamma v_{t-1} + \eta \frac{\partial \mathrm{E}}{\partial w_i}(w_{i,current} - \gamma v_{t-1})$$

$$w_{i,new} = w_{i,current} - v_t$$

$$\Downarrow$$

$$w_{i,new} = w_{i,current} - \left(\gamma v_{t-1} + \eta \frac{\partial \mathrm{E}}{\partial w_i}(w_{i,current} - \gamma v_{t-1})\right)$$

$$= w_{i,current} - \gamma v_{t-1} - \eta \frac{\partial \mathrm{E}}{\partial w_i}(w_{i,current} - \gamma v_{t-1})$$

Optimizer

이러한 방법을 learning rate decay라고 부름

- Adagrad (Adaptive Gradient)
  - SGD => 업데이트 횟수와 상관없이 learning rate를 동일하게
  - 하지만, Adagrad는 다르게
    - 지금까지 업데이트 된 정도를 반영한다.
    - 지금까지 업데이트가 많이된 parameter는 learning rate를 작게

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{G_{i,t} + \epsilon}} \frac{\partial \mathrm{E}}{\partial w_i}$$

지금까지 gradient의 합

$$\sum_{k=1}^{t} g_{i,k}^2$$

$g_{i,k}$는 k번째 업데이트에서 사용된 경삿값

ϵ is a smoothing term that avoids divisio

- 주요 문제
  - Gt가 갈수록 커진다 => 업데이트가 거의 발생하지 않는다.
  - Adadelta, RMSprop

Optimizer

- RMSprop (Root Mean Square Propagation)
  - Adagrad의 확장 버전
    - 무조건적으로 줄어드는 learning rate 문제를 보완
    - 합이 아니라 평균을 사용 [과거 gradients 들의 평균]

$$w_{i,t+1} = w_{i,t} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \frac{\partial \mathrm{E}}{\partial w_i}$$

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1 - \rho) g_t^2$$

이를 moving average라고 함
ρ는 보통 0.9 정도

Optimizer

- Adadelta
  - Adagrad 보완

$$RMS[g]_t = \sqrt{E[g^2]_t + \epsilon}$$

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$$

$$w_{i,t+1} = w_{i,t} - \frac{RMS[\Delta w_i]_{t-1}}{RMS[g]_t}\frac{\partial \mathrm{E}}{\partial w_i}$$

$\Delta w_{i,t} \Rightarrow$ t번째 업데이트에서의 가중치의 변화량

$$RMS[\Delta w_i]_{t-1} = \sqrt{E[\Delta {w_i}^2]_{t-1} + \epsilon}$$

- 경사 값만을 사용하는 것이 아니라 업데이트 정보도 같이 사용
- Adadelta는 초기 학습률의 값 설정 불필요 (아래와 같이 표현 가능, 조절도 가능)

$$w_{i,t+1} = w_{i,t} - \eta \cdot \frac{RMS[\Delta w_i]_{t-1}}{RMS[g]_{i,t}}\frac{\partial \mathrm{E}}{\partial w_i}, \qquad \text{where } \eta = 1$$

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Optimizer

- Adam
  - RMSprop (or Adadelta) + momentum
  - 다음과 같이 표현될 수 있음

$$w_{i,t+1} = w_{i,t} - \eta \frac{m_{i,t}}{\sqrt{v_{i,t} + \epsilon}}$$

$$m_{i,t} = \beta_1 \cdot m_{i,t-1} + (1 - \beta_1) \cdot g_{i,t}$$
$$v_{i,t} = \beta_2 \cdot v_{i,t-1} + (1 - \beta_2) \cdot g_{1,t}^2$$

$m_{i,t}$와 $v_{i,t}$의 값이 0으로 가까워지는 편향이 발생

$$\hat{m}_{i,t} = \frac{m_{i,t}}{1 - \beta_1^t}$$

$$\hat{v}_{i,t} = \frac{v_{i,t}}{1 - \beta_2^t}$$

$$0 < \beta_1, \beta_2 < 1$$
$$\beta_1 = 0.9,\ \beta_2 = 0.999$$

$$w_{i,t+1} = w_{i,t} - \eta \frac{\hat{m}_{i,t}}{\sqrt{\hat{v}_{i,t} + \epsilon}}$$

  - 자세한 내용은
    - Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
    - https://ruder.io/optimizing-gradient-descent/index.html#adam

Optimizer

- 참고: Nadam
  - Adam과 NAG을 결합한 방법
  - 일반 모멘텀이 아니라 Nesterov 모멘텀 사용

Optimizer

- Optimizer들의 비교
  - http://ruder.io/optimizing-gradient-descent/index.html#visualizationofalgorithms
- Which optimizer to use?
  - https://ruder.io/optimizing-gradient-descent/index.html#whichoptimizertouse

경사하강법

- 기타 용어들
  - 가중치 감쇠 (weight decay)
    - L2 규제화와 유사
    - 하지만, L2 규제화 방법과 달리, 가중치 감쇠 방법은 아래 공식을 이용해 학습되는 가중치의 절댓값을 줄이는 방법

$$w_{1,1,new} = (1-\lambda)\, w_{1,1,current} - \eta \frac{\partial \mathrm{E}}{\partial w_{1,1}}\left(w_{1,1,current}\right)$$

경사하강법

- 기타 용어들
  - 학습률 감쇠 (learning rate decay)
    - 옵티마이저가 자체적으로 학습률을 줄이는 방법이 아니라 사용자가 직접 업데이트마다 사용되는 학습률의 값을 줄이는 것을 의미
    - 예: 지수 감쇠 (Exponential decay)

$$\eta_t = \delta^{t/T} \cdot \eta_0$$

t/T : integer division

- η_0는 초기 학습률, η_t는 t 단계의 업데이트에서의 학습률을 의미
- T는 학습률 감쇠가 발생하는 업데이트 횟수, 예를 들어, $T = 1000$이라고 한다면 1,000번의 업데이트마다 한 번의 학습률 감쇠가 발생하는 것을 의미

    - Others
      - https://keras.io/api/optimizers/learning_rate_schedules/

Weight initialization

가중치 초기화

- Why initialize weights?
  - 가중치의 초기값이 적절하지 않으면 (너무 크거나, 너무 작으면) 학습이 제대로 진행되지 않고, 모형의 성능이 감소한다.
  - Example
    - 너무 큰 경우
      - 각 노드에 입력되는 값이 너무 커진다.
        - sigmoid, tanh 의 경우, 경사소실 문제 발생 ↑
        - 비용함수의 값이 커져, 경사값이 커지는 경향이 있음
    - 너무 작은 경우
      - 활성화 함수의 입력값이 0에 가깝게 되어, 활성화함수의 출력값이 특정한 값만을 가질 수도 있고 (sigmoid의 경우 0.5에 가까운 값들), 출력값이 0에 가까울수도 있다. ⇒ 모형의 성능이 좋지 않게 된다.
  - 주요 방법
    - Xavier, He 방법
    - 두 방법 모두 특정 노드에 입력되는 값들의 분산과 출력되는 값들의 분산의 크기를 동일하게 맞추는 것을 감안한 방법

가중치 초기화

- 주요 방법
  - 활성화함수: sigmoid or tanh 인 경우
    - Xavier 균등 초기화 방법

$$w_0 \sim U\left[-\sqrt{\frac{6}{n+m}}, \sqrt{\frac{6}{n+m}}\right]$$

    - Xavier 정규 초기화

$$w_0 \sim N\left(0, \frac{2}{n+m}\right)$$

$U[a,b]$의 분산은 $\frac{1}{12} \cdot (b-a)^2$

분산이 모두 동일

n은 이전 층에 존재하는 노드의 수, m은 현재 층의 노드의 수

Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249-256). JMLR Workshop and Conference Proceedings.

가중치 초기화

- 주요 방법 (cont'd)
  - 활성화함수: relu인 경우
    - He 정규 초기화 방법

$$w_0 \sim N\left(0, \frac{2}{n}\right)$$

    - He 균등 초기화 방법

n은 이전 층에 존재하는 노드의 수

$$w_0 \sim U\left[-\sqrt{6/n}, \sqrt{6/n}\right]$$

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).

가중치 초기화

- Keras의 경우
  - https://keras.io/api/layers/initializers/
  - 기본신경망의 경우, kernel_initializer='glorot_uniform' 로 설정되어 있음, 즉 Xavier 방법
  - bias_initializer="zeros"

Q & A