12강: Backprop & Improving Neural Network

3. Backpropagation

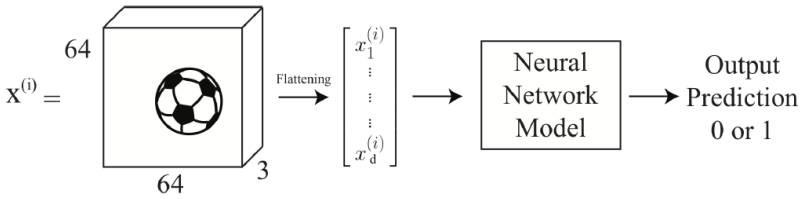
우리에게 이미지에 축구 공이 있는지 없는지 판단하는 문제가 있다고 생각해보자.

input image $x^{(i)}$이고 ouput은 공이 있을 때 1, 없을 때는 0인 binary 값을 갖는다.

- 참고

1. input vector 표현 방법

image는 픽셀 숫자에 대응하는 element 숫자로 구성된 matrix로 나타낼 수 있다. 그러나 color image는 volume으로 나타낸다. 칼라는 red, green, blue (RGB)로 표현되기 때문에 아래 그림처럼 image는 64 x 64 x 3 로 표현된다. 우리는 이것을 12,288개의 element를 포함한 single vector로 flatten하여 사용한다.



2. Neural Network model은 두개의 구성요소가 존재한다.

(i) the network architerture : layer가 몇 개인지, neuron이 몇 개인지, neuron들은 어떻게 연결되어 있는지,,,

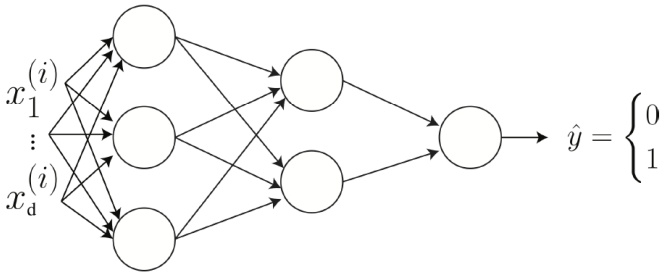
(ii) the parameters (weights)

이번 시간에는 어떻게 파라미터를 학습하는지 알아볼 것이다.

이는 3.1 parameter initialization, 3.2 optimization, 3.3 analyzing these parameter 순서로 알아볼 것이다.

3.1. Parameter Initalization

다음과 같이 2개의 layer를 가진 neural network가 있다고 해보자. input 은 flattened image vector이다. 여기서 첫번째 hidden layer를 보면 모든 input은 다음 layer의 모든 neuron과 연결되어 있기 때문에 fully connected layer라고 하는 것이다.



- 다음으로 우리는 이 network에 parameter가 얼마나 있는지 계산해야 한다.
  한 방법으로는 forward propagation을 직접 계산하는 것이다.

$$z^{[1]} = W^{[1]}x^{(i)} + b^{[1]} \tag{3.1}$$
$$a^{[1]} = g(z^{[1]}) \tag{3.2}$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \tag{3.3}$$
$$a^{[2]} = g(z^{[2]}) \tag{3.4}$$
$$z^{[3]} = W^{[3]}a^{[2]} + b^{[3]} \tag{3.5}$$
$$\hat{y}^{(i)} = a^{[3]} = g(z^{[3]}) \tag{3.6}$$

여기에서

$z^{[1]}, a^{[1]} \in R^{3\times 1}$ → 3차

$z^{[2]}, a^{[2]} \in R^{2\times 1}$ → 2차

$z^{[3]}, a^{[3]} \in R^{1\times 1}$ → 1차

임을 이미 알고 있다.

이제는 $W^{[1]}$의 size를 계산해야 한다.

input $x \in R^{d\times 1}$임을 알고 있기 때문에 우리가 알고 싶은 $W^{[1]}$의 size는 (3.7)처럼 나타낼 수 있다.

$$z^{[1]} = W^{[1]}x^{(i)} = \mathbb{R}^{3\times 1} \quad \text{Written as sizes:} \quad \mathbb{R}^{3\times 1} = \mathbb{R}^{?\times ?} \times \mathbb{R}^{d\times 1} \tag{3.7}$$

matrix 곱셈을 하기 위해서는, ? x ? 는 3 x d가 되어야 함을 알 수 있다.

또한 bias는 3 x 1이 되어야 한다. 그 이유는 $W^{[1]}x^{(i)}$의 size 3 x 1과 맞아야 하기 때문이다.

우리는 이 과정을 모든 layer에 반복해야 한다. 그 결과로 다음의 $W^{[2]}$, $W^{[3]}$결과를 얻을 수 있다.

$$W^{[2]} \in \mathbb{R}^{2\times 3}, b^{[2]} \in \mathbb{R}^{2\times 1} \quad \text{and} \quad W^{[3]} \in \mathbb{R}^{1\times 2}, b^{[3]} \in \mathbb{R}^{1\times 1} \tag{3.8}$$

최종적으로 첫번째 layer에는 3n+3, 두번째 layer에는 2 x 3 + 2, 세번째 layer에는 2+1 parameter가 존재하고 이를 합쳐서 총 3n+14 개의 파라미터가 주어진다.

- **training process**

1. **Parameter Initalization**

neural network를 학습시키기 전에 이 parameter에 대한 initial value를 선택해야 한다.

initial value로 zero를 사용하지 않는데 그 이유는 이럴 경우 첫번째 층의 ouput이 zero로 구성된 matrix형태로 나오면 나중에 파라미터를 업데이트하는데 문제를 일으키기 때문이다. (그라디언트가 항상 같다)

해결책은 랜덤하게 파라미터를 small value로 initialize하는 것이다.

이렇게 파라미터를 initialize하면 neural network를 gradient descent로 학습시키게 된다.

training process의 다음 step은 파라미터를 업데이트 하는 것이다.

- log loss :

$$L(\hat{y}, y) = -[(1-y)log(1-\hat{y}) + ylog\hat{y}]$$

log loss결과는 single scalar value이다. 이 $\mathcal{L}$ 값으로 neural network의 층에 있는 모든 파라미터를 업데이트시켜야 하는데 그 방법은 다음과 같다.

(any layer index l , α는 learning rate)

$$W^{[\ell]} = W^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial W^{[\ell]}} \tag{3.10}$$

$$b^{[\ell]} = b^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial b^{[\ell]}} \tag{3.11}$$

앞서 모든 파라미터를 zero로 설정하는 것은 안된다고 했는데 이제 그 이유를 이제 알아볼 것이다.

파라미터가 zero면 $W^{[3]}, b^{[3]}$이 0이기 때문에, $z^{[3]} = W^{[3]}a^{[2]} + b^{[3]} = 0$ 이다. 그러나 neural network의 output은 $a^{[3]} = g(z^{[3]})$ 일 때, $g(.)$가 sigmoid 함수라고 가정하면,

그 결과는 input $x^{(i)}$값에 상관없이 $a^{[3]} = g(0) = 0.5$으로 나온다.

만약 모든 파라미터가 모두 같은 (non-zero) 값이면 어떻게 될까?

이 경우에는 첫 번째 층의 활성화 함수를 살펴보자

$$a^{[1]} = g(z^{[1]}) = g(W^{[1]}x^{(i)} + b^{[1]}) \tag{3.12}$$

그 결과 $a^{[1]}$의 각 element값은 같을 것이다.($W^{[1]}$에 모두 같은 값을 가지고 있기 때문!) 이것은 모든 층에서도 같게 나타난다. 그래서 그라디언트를 계산하면 같은 층에 있는 뉴런들은 final loss에 동등하게 기여하게 된다 (뉴런이 같은 것을 학습하게 된다는 것!). 이 성질을 symmerty라고 한다.

따라서 random initialization 보다 더 나은 방법이 존재한다. 이 방법이 Xavier/ he initialization이다.

mini-normalization technique:

$$w^{[\ell]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[\ell]} + n^{[\ell-1]}}}\right) \tag{3.13}$$

$n^{[l]}$: l 층에 있는 뉴런의 숫자

한 층에 대해 input 분산을 $\sigma^{(in)}$, ouput의 분산을 $\sigma^{(out)}$라고 하면 Xavier/ He initialization은 $\sigma^{(in)}$을 $\sigma^{(out)}$와 유사하게 만든다고 한다. → ~~어떤원리인지 궁금하다,,,, 궁금해,,,,,~~

3.2 Optimization

2. **update parameters**

앞서 neural network의 prarmeter가 $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$라고 했다.

이것을 업데이트 하기 위해서는 위에서 보았던 update rule에

$$W^{[\ell]} = W^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial W^{[\ell]}} \tag{3.10}$$

$$b^{[\ell]} = b^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial b^{[\ell]}} \tag{3.11}$$

stochastic gradient descent (SGD)를 적용해야 한다.

- W3 그라디언트 계산

우리는 먼저 그라디언트를 나타내는 $W^{[3]}$ term 부터 계산해야 한다. 그 이유는 loss에서 $W^{[1]}$계산이 더 복잡하기 때문인데 간단하게 생각하면 $W^{[3]}$이 output $\hat{y}$에 가까이 있기 때문이다.

계산 과정은 다음과 같다.

$$\frac{\partial \mathcal{L}}{\partial W^{[3]}} = -\frac{\partial}{\partial W^{[3]}} \Big((1-y)\log(1-\hat{y}) + y \log \hat{y} \Big) \tag{3.14}$$

$$= -(1-y)\frac{\partial}{\partial W^{[3]}} \log \Big(1 - g(W^{[3]}a^{[2]} + b^{[3]}) \Big) \tag{3.15}$$

$$- y\frac{\partial}{\partial W^{[3]}} \log \Big(g(W^{[3]}a^{[2]} + b^{[3]}) \Big) \tag{3.16}$$

$$= -(1-y)\frac{1}{1 - g(W^{[3]}a^{[2]} + b^{[3]})}(-1)g'(W^{[3]}a^{[2]} + b^{[3]}){a^{[2]}}^T \tag{3.17}$$

$$- y\frac{1}{g(W^{[3]}a^{[2]} + b^{[3]})} g'(W^{[3]}a^{[2]} + b^{[3]}){a^{[2]}}^T \tag{3.18}$$

$$= (1-y)\sigma(W^{[3]}a^{[2]} + b^{[3]}){a^{[2]}}^T - y(1 - \sigma(W^{[3]}a^{[2]} + b^{[3]})){a^{[2]}}^T \tag{3.19}$$

$$= (1-y)a^{[3]}{a^{[2]}}^T - y(1 - a^{[3]}){a^{[2]}}^T \tag{3.20}$$

$$= (a^{[3]} - y){a^{[2]}}^T \tag{3.21}$$

- 위의 계산을 위해 알아두어야 할 것

1. 우리는 활성화함수로 sigmoid 함수를 사용했는데

시그모이드 함수의 미분 : $g' = \sigma' = \sigma(1-\sigma)$

2. $a^{[3]} = \sigma(W^{[3]}a^{[2]} + b^{[3]})$ 이라서 위의 $\hat{y}$에 대입해주게 된 것이다.

이를 통해서 최종적으로 $W^{[3]}$의 그라디언트를 계산하였다.

- W2 그라디언트 계산

이제 $W^{[2]}$의 gradient를 구해볼 것인데 위에처럼 복잡하게 $\partial\mathcal{L}/\partial W^{[2]}$를 계산하는 것이 아니라 chain rule를 이용하여 쉽게 계산해볼 것이다.

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{?}\frac{?}{\partial W^{[2]}} \tag{3.22}$$

이때 forward propagation을 보면 loss $\mathcal{L}$이 $\hat{y} = a^{[3]}$에 의존한다는 것을 알 수 있다. chain rule을 이용해서 $\partial a^{[3]}/\partial a^{[3]}$을 추가해보자.

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{?}\frac{?}{\partial W^{[2]}} \tag{3.23}$$

우리는 $a^{[3]}$이 직접적으로 $z^{[3]}$에 관련있다는 것을 알고 있다. 이를 추가해보자.

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}\frac{\partial z^{[3]}}{\partial a^{[2]}}\frac{\partial a^{[2]}}{?}\frac{?}{\partial W^{[2]}} \tag{3.25}$$

또한 $z^{[3]}$이 직접적으로 $a^{[2]}$에 관련있다는 것을 알고 있다.

이때 우리는 $W^{[2]}$ 혹은 $b^{[2]}$를 사용할 수 없는데 그 이유는 $a^{[2]}$는 (3.5), (3.6)에 등장하는 common element이기 때문이다. backpropagation에는 common element만 요구된다.

(참고)

$$z^{[3]} = W^{[3]}a^{[2]} + b^{[3]} \tag{3.5}$$
$$\hat{y}^{(i)} = a^{[3]} = g(z^{[3]}) \tag{3.6}$$

암튼 $a^{[2]}$를 추가하면 다음과 같다.

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}\frac{\partial z^{[3]}}{\partial a^{[2]}}\frac{\partial a^{[2]}}{?}\frac{?}{\partial W^{[2]}} \tag{3.25}$$

다시 $a^{[2]}$는 $z^{[2]}$에 의존하는데 이제 $z^{[2]}$는 $W^{[2]}$에 직접적으로 의존한다.

따라서 다음과 같이 최종적으로 chain을 완성할 수 있다.

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}\frac{\partial z^{[3]}}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial z^{[2]}}\frac{\partial z^{[2]}}{\partial W^{[2]}} \tag{3.26}$$

(recall : $\partial\mathcal{L}/\partial W^{[3]}$)

$$\frac{\partial \mathcal{L}}{\partial W^{[3]}} = (a^{[3]} - y)a^{[2]T} \tag{3.27}$$

우리는 처음에 $\partial \mathcal{L}/\partial W^{[3]}$를 계산했기 때문에 $a^{[2]} = \partial z^{[3]}/\partial W^{[3]}$, $(a^{[2]} - y) = \partial \mathcal{L}/\partial z^{[3]}$임을 알고 있다. 우리는 이 값들을 (3.26)식에 대입할 것이다. 그 결과는 다음과 같다.

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}}_{(a^{[3]}-y)}\underbrace{\frac{\partial z^{[3]}}{\partial a^{[2]}}}_{W^{[3]}}\underbrace{\frac{\partial a^{[2]}}{\partial z^{[2]}}}_{g'(z^{[2]})}\underbrace{\frac{\partial z^{[2]}}{\partial W^{[2]}}}_{a^{[1]}} = (a^{[3]} - y)W^{[3]}g'(z^{[2]})a^{[1]} \tag{3.28}$$

이렇게 간단하게 과정을 유도했지만 아직 끝난 것이 아니다. 왜냐면 matrix 계산을 위해 식을 변형해줘야 하기 때문이다.

모든 term의 dimension을 살펴보면 output shape를 맞추기 위해 순서를 바꾸고 마지막 term에는 transpose를 취해줘야 한다.

$$\underbrace{\frac{\partial \mathcal{L}}{\partial W^{[2]}}}_{2\times 3} = \underbrace{(a^{[3]} - y)}_{1\times 1}\underbrace{W^{[3]}}_{1\times 2}\underbrace{g'(z^{[2]})}_{2\times 1}\underbrace{a^{[1]}}_{3\times 1} \tag{3.29}$$

$$\underbrace{\frac{\partial \mathcal{L}}{\partial W^{[2]}}}_{2\times 3} = \underbrace{W^{[3]^T}}_{2\times 1} \circ \underbrace{g'(z^{[2]})}_{2\times 1}\underbrace{(a^{[3]} - y)}_{1\times 1}\underbrace{a^{[1]^T}}_{1\times 3} \tag{3.30}$$

이러한 방식으로 $\partial \mathcal{L}/\partial W^{[2]}$, $\partial \mathcal{L}/\partial W^{[3]}$을 활용하여 나머지 파라미터에 대해 그라디언트를 얻을 수 있다.

참고로 이렇게 backpropagartion과정에서 forward propagation에서 계산되었던 W, activation function, linear variables들이 필요하기 때문에 이 값들이 메모리에 저장되어 있어야 한다.

다시 optimization으로 돌아와 보자. 앞서서는 stochastic gradient descent를 언급했는데 이제 gradient descent에 대해 얘기해볼 것이다.

for any singer layer l, undate rule:

$$W^{[\ell]} = W^{[\ell]} - \alpha\frac{\partial J}{\partial W^{[\ell]}} \tag{3.31}$$

이때 J는 cost function으로

$J = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}^{(i)}$, $\mathcal{L}^{(i)}$은 single example에 대한 loss이다.

gradient descent가 stochastic gradient descent와 다른 점은 cost function J가 더 정확한 그라디언트를 얻는다. 대신 $\mathcal{L}^{(i)}$는 noisy하다.

stochastic gradient descent는 (full) gradeint descent로부터 대략적인 그라디언트를 얻는다.

gradient descent의 단점은 모든 example에 대해 모든 활성화함수를 계산하기 어렵하는 것이다.

- mini batch gradient descent

그래서 이번에는 이 둘의 중간 단계 느낌의 mini batch gradient descent를 사용할 것이다.

cost function J_{mb}는 다음과 같다.

$$J_{\mathrm{mb}} = \frac{1}{\mathrm{B}} \sum_{i=1}^{\mathrm{B}} \mathcal{L}^{(i)} \tag{3.32}$$

B: mini batch 안의 example 개수

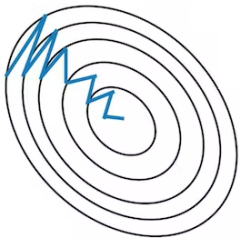
- GD+ momentum

이때 다른 optimization 방법으로 momentum를 추가할 수 있다. mini batch stochastic gradient를 보면 any single layer l에 대해 undate rule이 다음과 같다.

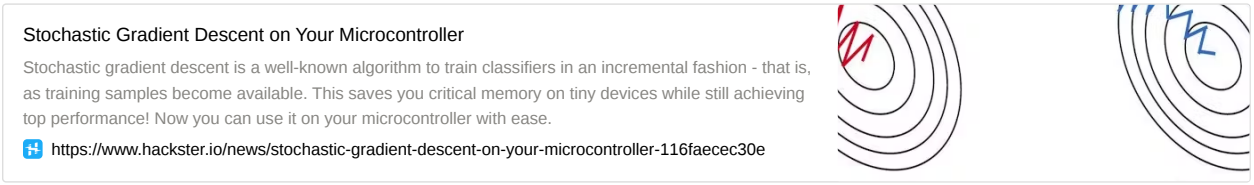
$$\begin{cases} v_{dW^{[\ell]}} = \beta v_{dW^{[\ell]}} + (1-\beta)\frac{\partial J}{\partial W^{[\ell]}} \\ W^{[\ell]} = W^{[\ell]} - \alpha v_{dW^{[\ell]}} \end{cases} \tag{3.33}$$

여기에는 두단계가 존재한다. 따라서 weight update는 이 update step에서의 cost J와 velocity $v_{dW^{[l]}}$에 의존한다. 여기서 β에 의해 중요도가 결정된다. 예를 들어 차 브레이크를 밟으면 가속도에 의해 앞으로 나아가는 것처럼 optimization을 할때 velocity $v_{dW^{[l]}}$는 끊임없이 그라디언트의 정보를 기록할 것이다. 그러면 상당히 학습할 때 도움을 준다고 한다.

Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum



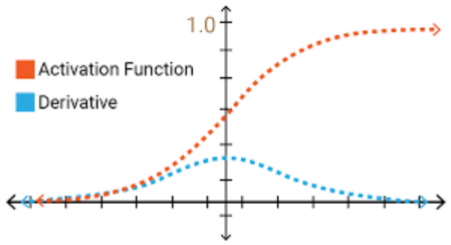
강의에서는 위의 방법들을 improving neural network라는 주제로 묶어서 설명하고 있다. 강의 내용을 한번 정리해보자..!

<Improving neural network>

1. **activation function**

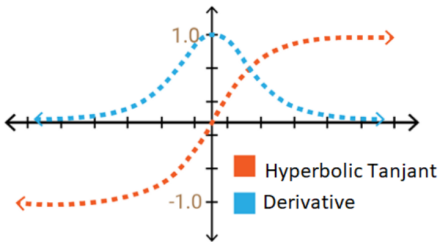
- sigmoid

Sigmoid Function	Output Range	Derivative
$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f(x) \in [0, 1]$	$f'(x) = f(x)(1 - f(x))$



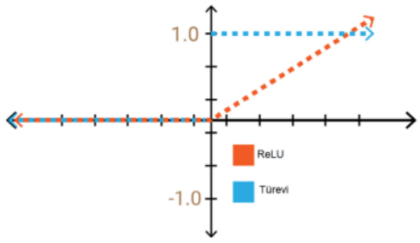
- tanh

Tanh Function	Output Range	Derivative
$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$[-1, 1]$	$f'(x) = 1 - f(x)^2$



- Relu

Relu Function	Output Range	Derivative
$f(x) = max(x, 0) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f(x) \in [0, +\infty]$	$f'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$



- activation fuction을 사용하는 이유?

선형 함수가 아니라 비선형 함수를 사용하는 이유는 딥러닝 모델의 레이어 층을 깊게 가져갈 수 있기 때문이다.

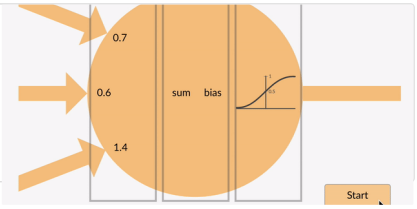
인공신경망에서 활성화 함수는 입력 데이터를 다음 레이어로 어떻게 출력하느냐를 결정하는 역할이기 때문에 매우 중요하다.

즉, 활성화 함수는 입력을 받아서 활성화 또는 비활성화를 결정하는 데에 사용되는 함수이다.

Activation Function 활성화 함수 - Sigmoid, tanh, ReLU, LeakyReLU

신경망 모델의 활성화 함수 기본 개념과 자주 사용되는 비선형 활성화 함수 Sigmoid, tanh, ReLU, LeakyReLU 특징과 활성화 함수 설정 가이드를 정리합니다. 활성화 함수의 역할과 특징 Activation function with respect to a single node 활성화 함수의 연산 과정을 하나의 node 뉴런에 대해서 정리하면 다음과 같다.

https://geniewishescometrue.tistory.com/entry/Activation-Function-%ED%99%9C%EC%84%B1%ED%99%94-%ED%95%A8%EC%88%98-Sigmoid-tanh-ReLU-Softmax



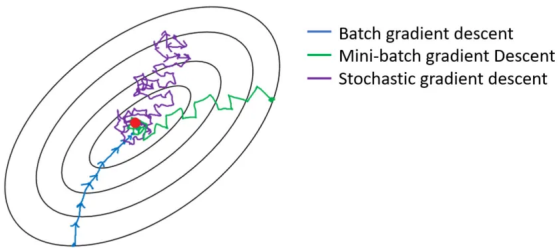
2. **initialization methods**

- normalizing your input: 평균으로 정규화/ 표준편차로 정규화 → 정규화시 학습이 더 쉬워진다.
- random initialization $W^{[l]} \sim \left(\frac{1}{n^{[l-1]}}\right)$ ← sigmoid사용시 잘 작동함
- xavier initailization $W^{[l]} \sim \sqrt{\frac{1}{n^{[l-1]}}}$ for tanh
- He initialization $W^{[l]} \sim \sqrt{\frac{2}{n^{[l]}+n^{[l-1]}}}$

전전딥딥 교안에 자세한 설명이 있다고 합니다…!

전전전전 선대 교안에도 그라디언트 설명이 잘 되어 있어요…! 연쇄법칙 설명도 있어요….!

3. **optimization**



- batch gradient descent
- stochastic descent
- mini batch descent

algorithm:

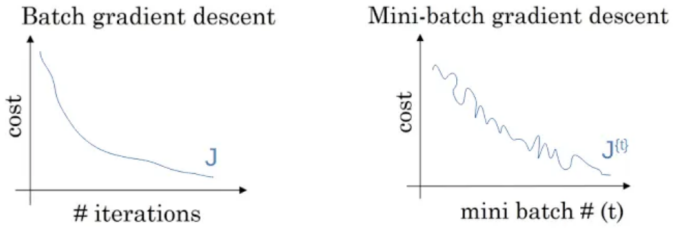
for iteration t=1,…

select batch $(x^{\{t\}}, y^{\{t\}})$

forward prop batch

backward prop batch

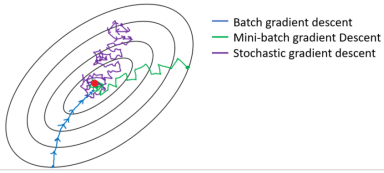
update $W^{[l]}, b^{[l]}$



Batch , Mini Batch and Stochastic gradient descent

There are different types of optimizer : Batch Gradient descent Mini batch Gradient descent Stochastic Gradient descent These types are nothing but different approach to send a data inside a network. BGD is a variation of the gradient descent algorithm that calculates the error for each eg in the training datasets, but

https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461



3.3 Analyzing the Paramaters

지금까지 우리는 파라미터를 initialize했고 그 파라미터를 optimize했다.

그 결과 training set에서는 96%의 정확도를 얻고 testing set에서는 64%의 정확도를 얻었다고 해보자.

몇 해결책으로는 1. collecting more data, 2. employing regularization. 3. making the model shallower이 있다.

간단하게 regularization technique에 대해 알아보자.

3.3.1 L2 Regularization

W: 모델의 모든 파라미터

L2 regularization을 추가한 cost function:

$$J_{L2} = J + \frac{\lambda}{2}||W||^2 \tag{3.34}$$

$$= J + \frac{\lambda}{2}\sum_{ij}|W_{ij}|^2 \tag{3.35}$$

$$= J + \frac{\lambda}{2}W^TW \tag{3.36}$$

J: standard cost function

λ : arbitrary value with a larger value indicating more regularization

W : contains all the weight matrices

L2 regularization의 update rule:

$$W = W - \alpha\frac{\partial J}{\partial W} - \alpha\frac{\lambda}{2}\frac{\partial W^TW}{\partial W} \tag{3.37}$$

$$= (1-\alpha\lambda)W - \alpha\frac{\partial J}{\partial W} \tag{3.38}$$

만약 gradient descent로 파라미터를 update한다면 $(1-\alpha\lambda)$ term은 필요없다. 즉 L2 regularization에서 모든 update에서 W에 대한 penalization을 포함하고 있다는 것이다. 이 penalization은 cost J를 증가시키는데 그 결과 개별 파라미터를 작게 만들어서 overfitting

을 예방한다.