2주차

이번 주 강의

[딥러닝 1단계] 2-2. 신경망과 로지스틱회귀 (계산 그래프~m개 샘플의 경사하강법), 3-1. 파이썬과 벡터화 (벡터화~로지스틱 회귀의 경사 계산을 벡터화 하기)

Computing derivatives

- 얼마나 x값을 변화시켜야 궁극적인 결과값이 원하는 만큼 변화할 수 있을까를 계산하기 위해서 dy/dx 를 구하고 싶은 것.
- 수식으로 바로 dy/dx 를 구할 수 없으니 미적분에서 Chain rule을 사용

  $\frac{dy}{da} * \frac{da}{dx}$

역전파(Backpropagation)

- 앞 뒤의 미분 했을 때 값들만 알면 되기에 긴 수식을 다 데리고 가지 않아도 된다는 장점이 있다.

로지스틱 회귀의 경사하강법

- $z = w^T x + b$
- $\hat{y} = a = \sigma(z)$ → $\frac{da}{dz} = a(1-a)$
- $\mathcal{L}(a, y) = -(y \log(a) + (1-y) \log(1-a))$ → $\frac{dL}{da} = \frac{-y}{a} + \frac{1-y}{1-a}$

→ $\frac{dL}{dz} = \frac{dL}{da} \times \frac{da}{dz} = a - y$

- 결론적으로 구하고 싶은 $\frac{dL}{dw}$과 $\frac{dL}{db}$ 를 Chain rule을 통해서 구하면 아래와 같다.

$$\frac{dL}{dw} = x \times (a-y), \quad \frac{dL}{db} = a - y$$

- 변수가 많아진다면?

→ 처음에 곱해지는 값만 바뀔 뿐 뒤의 $\frac{dL}{dz}$ 구하는 것은 바뀌지 않음.

m 개의 훈련 샘플에 대한 경사 하강

- 훈련 샘플이 많아졌다면 평균을 내주면 된다!

$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}(\mathcal{L}(a^{(i)}, y^{(i)}))$

- 변수가 많아졌다면?

→ 동일하게 알고 싶은 미지수의 변화만 봐주면 된다.

- 코드로 경사 하강 식 작성하기

$J = 0, \quad dw_1 = 0, \quad dw_2 = 0, \quad db = 0$

$\text{For } i = 1 \text{ to } m$

$$
\begin{aligned}
z^{(i)} &= w^T x^{(i)} + b \\
a^{(i)} &= \sigma(z^{(i)}) \\
J_{\text{temp}} &= -\Big(y^{(i)} \log(a^{(i)}) + (1 - y^{(i)}) \log(1 - a^{(i)})\Big) \\
dz^{(i)} &= a^{(i)} - y^{(i)} \\
dw_1 &+= x_1^{(i)} dz^{(i)} \\
dw_2 &+= x_2^{(i)} dz^{(i)} \\
db &+= dz^{(i)}
\end{aligned}
$$

$J = \frac{J}{m}, \quad dw_1 = \frac{dw_1}{m}, \quad dw_2 = \frac{dw_2}{m}, \quad db = \frac{db}{m}.$

문제점: for 문을 반복하면서 계산량이 너무 많아짐 → 벡터화를 통해 for 문을 없애

벡터화

- 벡터화란 무엇인가

→ 벡터화를 사용할 경우, for 문을 사용해서 여러번 곱하기를 해왔던 작업을 dot 연산을 통해 한번에 하는 것이 가능.(예를 들어 w와x 가 n 차원이라고 할 때 for 문을 n 번 반복해야 하지만, 벡터화는 1번에 가능함.)

```
import numpy as np
import time

a=np.random.rand(1000000)
b=np.random.rand(1000000)
tic=time.time() # 시작 시간
c=np.dot(a,b)
toc=time.time() #끝 시간

print("Vectorized version:"+str(1000*(toc-tic))+"ms")

c=0
tic=time.time()
for i in range(1000000):
   c+=a[i]+b[i]
toc=time.time()

print("For loop:"+str(1000*(toc-tic))+"ms")
```

→ 벡터화를 하게 될 경우, 병렬화의 장점을 사용할 수 있게 됨.

더 많은 벡터화 예제

- 벡터가 나온다고 해서 무조건 벡터화 연산을 한 것이 아님. for문을 벡터를 통해 줄여나가야 함.

```
J = 0, dw1 = 0, dw2 = 0, db = 0

for i in range(1, m):
    z_i = np.dot(w.T, x[i]) + b
    a_i = sigma(z_i)

    J += -(y[i] * np.log(a_i) + (1 - y[i]) * np.log(1 - a_i))
```

```
    dz_i = a_i - y[i]

    dw1 += x[1][i] * dz_i
    dw2 += x[2][i] * dz_i
    db += dz_i

J /= m, dw1 /= m, dw2 /= m, db /= m
```

w1, w2, ... 여러 개인 것을 한번에 처리하기 위해서 dw1과 dw2를 벡터로 만들기

`dw=np.zeros((n_x,1))` 로 처음에 정의해주기

로지스틱 회귀의 벡터화

1-m개의 훈련세트들을 for문을 도는 것이 아니라 벡터로 가로로 일렬로 만들어서 한번에 계산해주기

`Z=np.dot(w.T, X)+b` 를 사용하면 1~m의 z들을 한번에 모아서 계산할 수 있음

→ 파이썬 특징-브로드캐스팅: (1,m)크기의 행렬과 상수b를 더하기에 오류가 날 것 같지만, 자동으로 상수 b를 (1,m)크기로 맞춰줌

(벡터화 해서 한번에 계산할 수 있는 이유가 모든 훈련 세트에 대해서 같은 계산을 적용할 것이기에 통째로 만들어서 한번에 계산한다고 이해하기.)

로지스틱 회귀의 경사계산을 벡터화 하기

```
Z = np.dot(w.T, X) + b
A = sigma(Z)  # sigma는 활성화 함수, 예: sigmoid

dZ = A - Y
dW = (1 / m) * np.dot(X, dZ.T) #dot을 하게 되면 자동으로 더해짐!
db = (1 / m) * np.sum(dZ)

alpha = 0.01  # 학습률
```

```
w -= alpha * dW
b -= alpha * db
```