



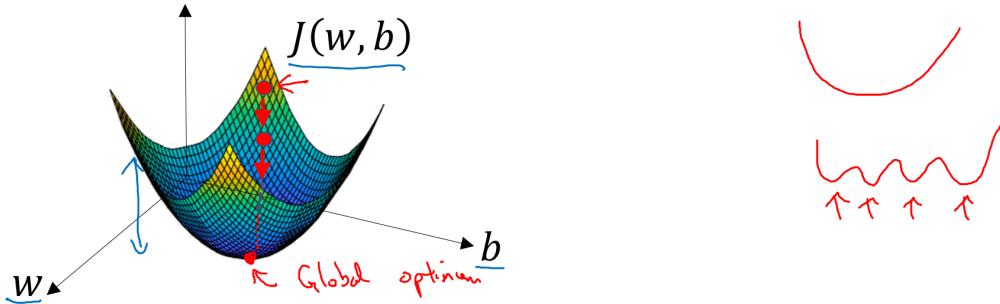
[2주차] Logistic Regression as a Neural Network

2주차

Gradient Descent

Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$ ↪
 $J(w, b) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$

Want to find w, b that minimize $J(w, b)$



Andrew Ng

2번째 줄에는 J 라는 비용함수가 있습니다. 이 함수는 파라미터 w 와 b 에 대한 함수입니다. 그리고 이것은 평균값으로 정의되죠.

그러므로 1 나누기 m 곱하기 이 loss함수의 합입니다. 그렇게해서 loss함수는 각각의 트레이닝 example에 대해서 알고리즘에 결과값 $\hat{y}(i)$ 이 얼마나 잘 쌓거나 ground true label $y(i)$ 과 얼마나 잘 비교되는지 측정합니다.

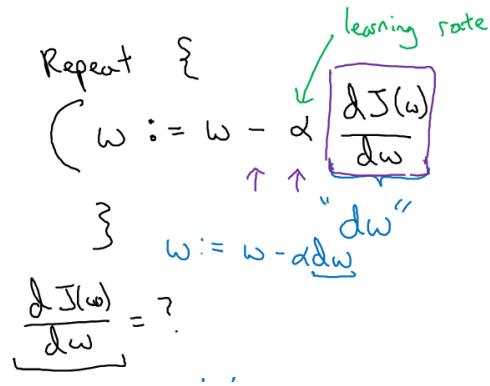
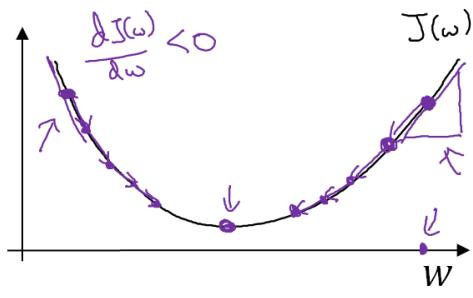
전체 공식은 오른쪽에 펼쳐져있습니다. 자 그래서, 비용함수는 트레이닝세트에서 파라미터 w 와 b 가 얼마나 잘 하는지를 측정하는데요 w 와 b 라는 파라미터를 배우기 위해서는, w 와 b 가 비용함수 $J(w, b)$ 를 최소화시키는 값을 찾는게 자연스러운 단계이겠죠?

이것이 gradient descent의 모습입니다. 이 표에서는 가로축이 공간 매개 변수 w 와 b 를 나타냅니다. 실제로 w 는 다른 다이멘션일 수 있지만 표에 나타내기 위해 w 와 b 가 실수라고하겠습니다. $J(w, b)$ 비용함수는 그러면 이 가로축 w 와 b 표면 어딘가에 있을 것입니다. 표면의 높이는 특정지점에서 $J(w, b)$ 의 값을 가르키는데요 저희는 J 라는 비용함수가 최소값이 되는 w 와 b 의 값을 구해야 합니다.

알고보면, 이 J 비용함수는 볼록함수입니다. 하나의 그릇 모양이라고 할 수 있는데요, 이것이 볼록함수이고, 이런 여러개의 local이 있는 비볼록 함수와는 다르죠. 이렇게 정의된 비용함수 $J(w, b)$ 는 볼록하고, 그렇기 때문에 이와 같은 비용함수 J 를 이용하는 것입니다. 로지스틱 회귀분석에서 말이죠. 파라티터의 적합한 값을 찾으려면, w 와 b 를 최소값으로 초기화해야 합니다. 이 빨간 점이 상징하듯이 말이죠. 로지스틱 회귀분석에서는 거의 모든 초기화 방법이 잘 구현됩니다. 일반적으로 0으로 초기화시키는데요, 무작위로 초기화 시키는 방법도 효율적입니다. 그렇지만 대부분의 사람은 보통 이러한 방법은 잘 사용하지 않죠.

이 함수가 볼록함수이기 때문에 어느 지점에서 초기화하더라도 똑같은 지점에 도달하거나 거의 비슷한 점에 도달할 것입니다.

Gradient Descent



이 그림은 기울기 강하 알고리즘을 나타내고 있습니다. 조금 더 쉽게 나타내기 위해 $J(w)$ 라는 함수가 있다고 해보겠습니다. 이 값을 최소화시키고 싶은데요, 이렇게 생겼다고 해보겠습니다. 일단은 b 를 무시하고, 다차원이 아닌 1차원적인 표로 만들겠습니다. 기울기 강하가 이렇게 해서, 해당 update를 계속 반복적으로 진행할 것입니다. w 값을 갖고 업데이트할 것입니다. 물론 표시로 w 를 업데이트함을 표시할텐데요, w 를 w 빼기 알파로하고, 이것은 derivative $dJ(w)/dw$ 입니다. 이것을 알고리즘이 합쳐질때까지 반복하겠습니다.

알아둘 점은, 여기 **알파는 학습 속도를 뜻하고, 한번에 얼마만큼 기울기 강하를 진행할 수 있는지 조절해줍니다.**

2번째로, 여기 이 값은 **derivative입니다.** 이것은 업데이트 또는 w 에 얼마나 변화를 줄지 여부를 알려주는 값입니다. 기울기 강하를 도입하기 위한 코드를 만들기 시작하면 dw 라는 변수가 derivative를 나타내도록 규칙을 만들 것입니다. 그렇게해서 코드를 만드는 경우, w : 는 w 빼기 알파 곱하기 dw 라고 적어볼 텐데요 dw 를 변수 이름으로 만들어서 이 derivative 항을 뜻하게 만듭니다.

자 이제 해당 기울기 강하 업데이트가 말이 되도록 만들겠습니다. w 가 여기 있었다고 하죠. 그러면 비용함수 $J(w)$ 에서 이 지점에 있습니다. **기억할 것은, derivative의 정의가 특정 지점에서 함수의 기울기라는 것입니다.** 함수의 기울기는 높이 나누기 너비, 맞죠? 이 작은 삼각형에서 말이죠, 기울기는 $J(w)$ 에서 이 탄젠트 지점이 됩니다. 그렇게해서, derivative는 양수입니다. W 는 w 빼기 학습 속도 곱하기 derivative 값만큼 업데이트 되는 것입니다.

derivative값은 양수이고, 그렇기 때문에 w 에서 빼게되는 것인데요 그러므로 왼쪽으로 진행합니다. 만약 이 큰 w 값으로 시작했다고하면 기울기 강하는 알고리즘이 천천히 파라티터를 줄이도록 할 것입니다. 다른 예제로, 만약 w 가 여기 있었다고 하면 이 지점에서는 dJ/dw 의 기울기는 음수일 것입니다. 이 경우, 기울기 강하 업데이트는 알파 곱하기 음수의 값을 뺄 것입니다. 결과적으로 w 를 천천히 증가시킬텐데요, w 를, 기울기 강하의 반복을 통해 점점 더 크게 만드는 효과가 있습니다. 그러므로 왼쪽에서 초기화를 진행하던 오른쪽에서 진행하던, 기울기 강하는 이 전역 최소값을 향해 움직일 것입니다. 여러분이 derivative이나 미적분학에 익숙해 있지 않으신 경우이거나, $dJ(w)/dw$ 가 뜻하는 바가 무엇인지 잘 모르시더라도 너무 걱정하지 마십시오.

오로지 w 가 파라티터인 경우, $J(s)$ 에 대한 기울기 강하를 적었습니다. **로지스틱 회귀분석에서는 비용함수가 w 와 b 라는 파라티터에 대한 함수입니다.** 이 경우, 기울기 강하의 inner loop는, 이와 같이 생긴 것인데요, 반복해야하는 이 것은 아래와 같습니다.

$$J(w, b)$$

$$w := w - \alpha \frac{\partial J(w, b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b}$$

$$\frac{\partial J(w, b)}{\partial w}$$

$$\frac{\partial J(w, b)}{\partial b}$$

"partial derivative" J

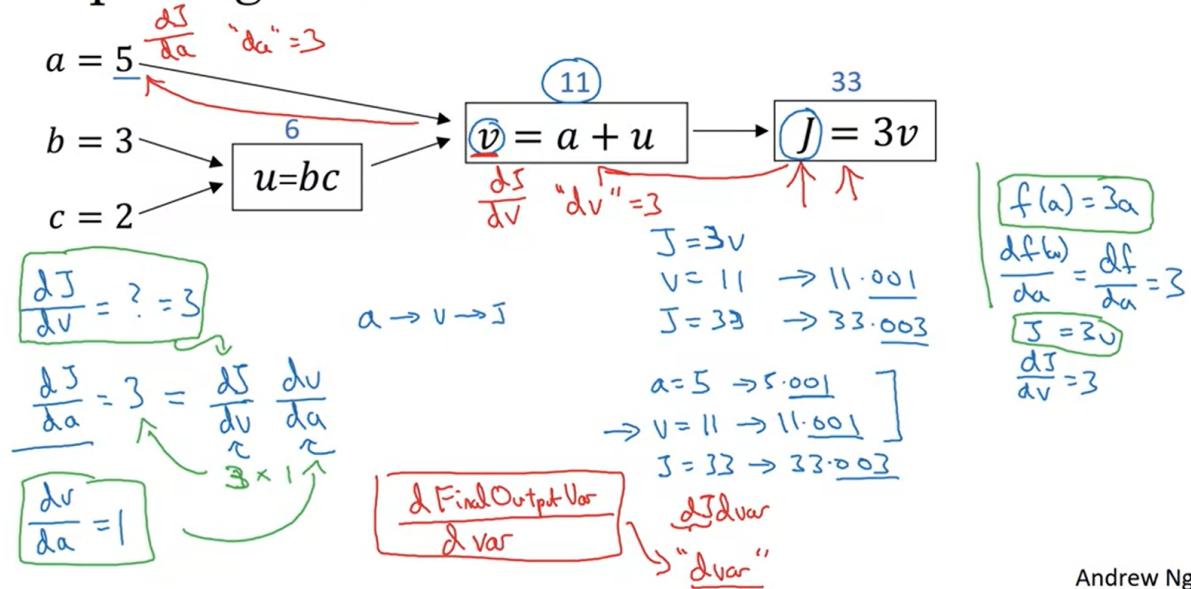
Andrew Ng

w 를 w 빼기 학습 속도 곱하기 w 에 대한 $J(w, b)$ derivative로 업데이트 합니다. b 는 b 빼기 학습속도 곱하기 b 에 대한 비용함수의 derivative로 b 를 업데이트 합니다. 그래서 이 밑에 있는 2개의 공식이 실제로 도입하는 업데이트입니다.

이 심볼은 소문자 d 이구요 조금 특화된 글꼴로 쓰는데 이러한 표기를 보면, $J(w, b)$ 의 기울기라는 것입니다. w 방향으로 $J(w, b)$ 함수가 얼마나 기울고 있는지를 나타냅니다.

2개 이상의 변수를 가지있는 경우, 이 partial derivative 심볼을 쓰고, 1개의 변수가 있는 경우, 소문자 d 를 사용합니다. partial derivative 심볼을 보시면 단순히 함수의 기울기가 이 변수들 중 하나에 대하여 계산된다고 생각하면 됩니다. 비슷하게, 앞서 다룬 미적분학 표기법을 동일하게 적용합니다, 그 이유는 여기서는 J 가 2개의 입력값이 있기 때문이죠. 이 밑에 있는 부분은 이 partial derivative 심볼과 같이 쓰여야 합니다. 이 것은 거의 똑같은데요, 소문자 d 가 의미하는 것과 거의 비슷합니다. 마지막으로 이것을 코드에 도입시킬 때는, 이 양이, 즉 w 를 업데이트하는 정도의 양을, dw 변수로 코드에서는 표기될텐데요 이 값 맞죠? b 를 업데이트하고 싶은 정도는 코드에서의 db 변수로 표기될 것입니다. 자 이렇게 gradient descent를 도입할 수 있는 것입니다. 만약 여러분이 미적분학을 수년동안 접하지 않으셨다면, 지금 본인의 현재 편안한 정도의 레벨보다 미적분학에서 더 많은 derivative를 다룬다고 생각하실텐데요, 만약 여러분이 이렇게 느끼시면 걱정하지 마십시오. 다음 비디오에서는 derivative에 대한 조금 더 직관적인 이해를 돋도록 하겠습니다. 미적분학에 대한 깊은 수학적인 지식없이도 직관적인 미적분학의 이해만으로도 신경망을 조금 더 효율적으로 만들 수 있을 것입니다. 다음 비디오로 넘어가서 derivative에 대해 조금 더 이야기해보도록 하겠습니다.

Computing derivatives

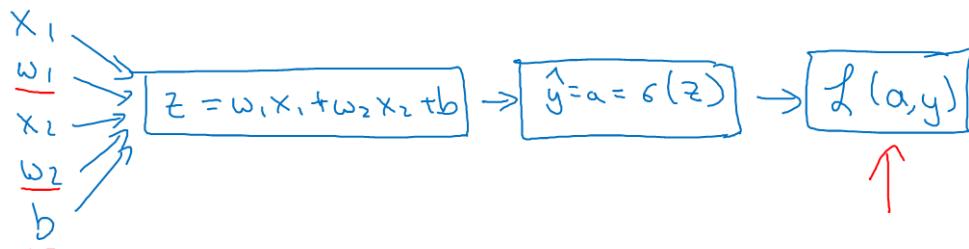


Andrew Ng

dFinalOutputVar/dvar와 같이 말이죠. 하지만 이것은 너무 긴 변수 이름이겠죠. 그냥 dvar이라는 변수 이름을 사용하겠습니다. 여기 이 양을 나타내기 위해서 말이죠. 그러므로 코드에서 쓰는 dvar는 J와 같은 최종 결과값 변수의 derivative를 나타낼 것입니다.

Logistic regression recap

$$\begin{aligned}\rightarrow z &= w^T x + b \\ \rightarrow \hat{y} &= a = \sigma(z) \\ \rightarrow \mathcal{L}(a, y) &= -(y \log(a) + (1 - y) \log(1 - a))\end{aligned}$$

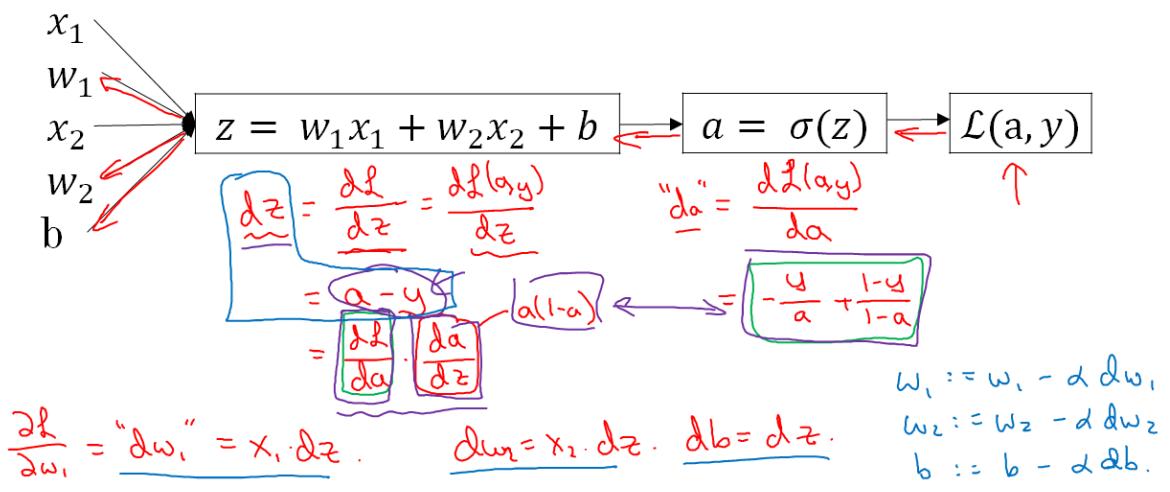


Andrew Ng

그러면 이제 로지스틱 회귀분석을 위한 기울기 강하에 대해 알아보겠습니다. 복습하자면 이전에 로지스틱 회귀분석을 이렇게 설정했었는데요, 예측 값 \hat{y} 은 이렇게 정의되었는데요, z 는 이것이구요, 일단 1가지 example에 초점을 맞추면, loss는 여기 example에 대해서는 이렇게 정의됩니다. a 가 로지스틱 회귀분석의 결과값이고 y 는 ground truth 레이블입니다.

로지스틱 회귀분석에서는, w 와 b 의 파라미터를 변형시킬 것입니다. 여기 loss 값을 줄이기 위해서 말이죠.

Logistic regression derivatives



Andrew Ng

[Step1]

여기 a 변수에 대해서 산출해야 합니다. 그렇기 때문에 여기 코드에서 da 를 이용해 변수를 표기하는데요, 만약 여러분이 미적분학과 익숙하면 이것이 $-y$ 나누기 a 더하기 $(1-y)$ 나누기 $(1-a)$ 라는 것을 보여줄 수 있습니다.

[Step2]

이제 여러분은 dz 를 보여줄 수 있는데요, 이 부분은 파이썬 코드의 변수이름인데요, 여기는 이제 loss의 derivative가 될 것입니다. z 나 l 에 대해서 말이죠. 아니면 a 와 y 가 포함된 파라미터를 이용하여 loss를 나타낼 수 있습니다. 맞죠? 둘 중의 아무 표기 방식이나 상관 없습니다. 그러면 이것이 $a-y$ 라는 것을 보여줄 수 있습니다.

$$\frac{dL}{dz} = a - y$$

[Step3]

그리고 여기 2개의 것들을 가지고 곱해줍니다. 그렇게 되면 이 공식들은 a 빼기 y 로 간략해지죠.

[Step4]

마지막 단계는 w 와 b 를 얼마나 바꿔야 하는지 계산하는 부분입니다. 특히, w_1 에 대한 derivative가 x_1 곱하기 dz 라는 것을 보여줄 수 있는데요, 코딩에서는 이것을 dw_1 이라고 합니다. 그리고 비슷한 방법으로, dw_2 , 즉 w_2 를 얼마나 바꿔야하는지 알려주는 수치는 x_2 곱하기 dz 와 b 입니다. 아 죄송합니다. b 가 아니라 db 죠, 이 값은 dz 와 동일합니다. 여러분이 만약 1가지 example에 대한 기울기 강하를 원할 경우, 이렇게 하면 됩니다. 여기 공식을 사용해서 dz 를 계산합니다. 그리고 여기 이 공식들을 사용해서 dw_1 , dw_2 , db 그리고 여기 update를 실행합니다. w_1 은 w_1 빼기 러닝속도 알파 곱하기 dw_1 입니다. w_2 도 이와 같이 업데이트 될텐데요, 그리고 B 는 B 빼기 러닝속도 곱하기 DB로 설정됩니다. 그러면 1개의 example의 경우를 보여주는 단계입니다.

이전 비디오에서는 1개의 트레이닝 example에 대해서 로지스틱 회귀분석을 위한 기울기 강하를 도입하기 위해 derivative를 산출하는 방법을 배웠는데요, 이제는 m개의 트레이닝 example에 대하여 구하고 싶습니다.

Logistic regression on m examples

$$J=0; \Delta w_1=0; \Delta w_2=0; \Delta b=0$$

For $i=1$ to m

$$z^{(i)} = \omega^\top x^{(i)} + b$$

$$\alpha^{(i)} = \sigma(z^{(i)})$$

$$J += -[y^{(i)} \log \alpha^{(i)} + (1-y^{(i)}) \log (1-\alpha^{(i)})]$$

$$\Delta z^{(i)} = \alpha^{(i)} - y^{(i)}$$

$$\Delta w_1 += x_1^{(i)} \Delta z^{(i)}$$

$$\Delta w_2 += x_2^{(i)} \Delta z^{(i)}$$

$$\Delta b += \Delta z^{(i)}$$

$\downarrow n=2$

[For Loop]

이제 여러분이 할 수 있는 것은 이런데요, $J=0$ 으로 초기화 하고, $dw1 = 0$, $dw2 = 0$, $db = 0$ 으로 설정하고, 이제 할 것은 이런 트레이닝 세트에 for loop를 사용해서 각각의 트레이닝 example에 대해서 derivative 값을 구한 뒤에, 그 값을 더할 것입니다. 이제 이렇게 하는 데요, i 가 1에서 m 까지, m 은 트레이닝 example의 개수입니다. z_i 는 w transpose x_i 더하기 b 이고요, 예측 값 a_i 는 z_i 의 시그마 그리고 j 를 더해보도록 하죠. j 플러스는 $y_i \log a_i$ 더하기 1 빼기 y_i 의 로그 1 빼기 a_i 하고 나서 전체 괄호 바깥으로 마이너스 부호를 앞에 두겠습니다. 그리고 이전에도 봤듯이 dzi 는 a_i 빼기 y_i 이고요 dw 플러스 값은 $x_1 i$ 곱하기 $d z_i$ 입니다. $dw2$ 플러스는 $x_2 i$ $d z_i$ 입니다. 아 그리고 저는 이 계산들을 오로지 2개의 특성만 있다는 가정하에 진행하고 있는 것입니다. 즉, $n=2$ 인 경우이죠, 안 그러면, 아니면 이것을 $dw1$, $dw2$, $dw3$ 등등 이어서 하겠죠, 그러면 db 플러스는 $d z_i$ 인데요, 이렇게 하면 for loop의 끝입니다.

$$J / = m$$
$$\Delta w_1 / = m; \Delta w_2 / = m; \Delta b / = m.$$

$\uparrow \quad \uparrow \quad \uparrow$

이렇게 m트레이닝 example에 대해서 모든 것을 마치면, 아직도 M으로 나누어야 할 텐데요, 평균을 구하는 것이기 때문입니다. 그러므로 dw1을 나눈 값이 m이고, dw2를 나누면 m이고, db 나누면 m이 되겠죠. 모든 평균에서 말이죠 그러면 전체 모든 계산에서 3개의 parameter w1, w2, b에 대한 j 비용함수의 derivative를 계산했는데요

$$\delta \omega_1 = \frac{\partial J}{\partial \omega_1}$$

계산 이후에는 dw1이 전체 비용함수의 derivative와 일치 하다는 것을 알 수 있습니다. w1에 대해서 말이죠. 비슷하게 dw2와 db에서도 말이죠. 보시면 알겠지만 dw1과 dw2는 위 첨자 i가 없습니다. 그 이유는 여기 코드에서 이것을 누산기들로 이용해서 전체 트레이닝 세트에 대해 사용하기 때문입니다. 반면에, dxi는 여기서 1개의 트레이닝 샘플에 대한 dz 값이었는데요, 그렇기 때문에 1개의 트레이닝 샘플임을 나타내기 위해 위 첨자 i가 있었던 것인데요

$$\begin{aligned} \omega_1 &:= \omega_1 - \alpha \underline{\delta \omega_1} \\ \omega_2 &:= \omega_2 - \alpha \underline{\delta \omega_2} \\ b &:= b - \alpha \underline{\delta b} \end{aligned}$$

기울기 강하의 1개의 단계를 도입하기 위해서는 w1을 도입하고, 이것은 w1 빼기 러닝속도 곱하기 dw1으로 업데이트 되고요. dw2는 w2 빼기 러닝속도 곱하기 dw2로 업데이트 됩니다. 그리고 b는 b 빼기 러닝속도 곱하기 db로 업데이트됩니다. 그리고 dw1, dw2, db는 이전에

왼쪽에서 계산했던 대로의 값입니다. 마지막으로 여기 J 는 비용함수의 올바른 값일 것입니다.
그러면 이제 슬라이드의 모든 내용은 1가지 단계의 기울기 강하를 도입하는 것인데요, 여기 슬라이드에 있는 모든 내용을 복수로 반복해야 복수의 gradient descent 단계를 갖출 수 있습니다.

Logistic regression derivatives

```

 $J = 0, dw1 = 0, dw2 = 0, db = 0$ 
→ for i = 1 to m:
     $z^{(i)} = w^T x^{(i)} + b$ 
     $a^{(i)} = \sigma(z^{(i)})$ 
     $J += -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$ 
     $dz^{(i)} = a^{(i)}(1 - a^{(i)})$ 
    for j=1..nx
         $dw_1 += x_1^{(i)} dz^{(i)}$ 
         $dw_2 += x_2^{(i)} dz^{(i)}$ 
         $db += dz^{(i)}$ 
 $J = J/m, dw1 = dw1/m, dw2 = dw2/m, db = db/m$ 

```

Andrew Ng

이건 로지스틱 회귀분석법에서 derivative를 계산하는 코드인데요. 여기서는 2개의 for-loop가 있었습니다. 하나는 여기 위에 있는 것이고, 2번째 것은 이것입니다. 예제에서는, n_x 가 2인데요, 2개의 특성보다 더 많은 경우, for-loop가 $dw1$, $dw2$, $dw3$ 등등에 있어야 할 것입니다. 그렇기 때문에 마치 j 는 1, 2, 그리고 n_x 인 것처럼 dWj 가 업데이트 됩니다.

Logistic regression derivatives

$$\begin{aligned}
 J &= 0, \quad \boxed{\text{dw1} = 0, \text{dw2} = 0}, \quad db = 0 \quad dw = np.zeros((n_x, 1)) \\
 \rightarrow \text{for } i &= 1 \text{ to } m: \\
 z^{(i)} &= w^T x^{(i)} + b \\
 a^{(i)} &= \sigma(z^{(i)}) \\
 J &+= -[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \\
 \downarrow \text{for } j &= 1 \dots n_x \\
 dz^{(i)} &= a^{(i)}(1 - a^{(i)}) \\
 \boxed{\frac{dw_1}{dw_2}} &+= \boxed{x_1^{(i)} dz^{(i)} \quad x_2^{(i)} dz^{(i)}} \quad n_x = 2 \quad dw += x^{(i)} dz^{(i)} \\
 db &+= dz^{(i)} \\
 J &= J/m, \quad \boxed{dw_1 = dw_1/m, \quad dw_2 = dw_2/m}, \quad db = db/m \\
 dw /&= m.
 \end{aligned}$$

Andrew Ng

우리는 이 2번째 for-loop를 제거하고 싶은데요 하는 방법은, 일단은 대놓고 dw1, dw2 등을 0으로 초기화시키기 보다는

이것을 먼저 제거하고 dw를 벡터로 만들겠습니다. 그러면 dw를 np.zeros가 되게 만들고 이 nx를 1차원 벡터로 만들겠습니다. 그러면 여기서 개인 요소에 대한 for-loop 대신에 그러면 여기서 개인 요소에 대한 for-loop 대신에 이 벡터 값 방법을 사용하겠습니다. dw 플러스는 xi 곱하기 dz(i) 말이죠. 그리고 마지막으로, **이것 대신에 dw 나누기가 m이 되도록 하겠습니다.** 이렇게 해서 for-loop가 2개에서 1개가 되었는데요

Implementing Logistic Regression

```

J = 0, dw1 = 0, dw2 = 0, db = 0
for i = 1 to m:
    z(i) = wTx(i) + b ←
    a(i) = σ(z(i)) ←
    J += -[y(i) log a(i) + (1 - y(i)) log(1 - a(i))]
    dz(i) = a(i) - y(i) ←
    { dw1 += x1(i)dz(i)
      dw2 += x2(i)dz(i)
      db += dz(i) } dw += x(i)*dz(i)
J = J/m, dw1 = dw1/m, dw2 = dw2/m
db = db/m

```

```

for iter in range(1000): ←
    z = wTX + b
    = np.dot(w.T, X) + b
    A = σ(z)
    dZ = A - Y
    dw = 1/m * X * dZT
    db = 1/m * np.sum(dZ)
    w := w - αdw
    b := b - αdb

```

Andrew Ng

dw1, dw2부분은 이전 도입에 있었던 내용인데요, 이 1개의 for loop은 이미 없앤 상태입니다. 이제 적어도 dw은 벡터이고 개별적으로 dw1, dw2, 등등을 업데이트 해줬습니다.

이제 바깥쪽 for loop를 없애기 위해 오른쪽과 같은 작업을 진행한다. 그러면 여러분은 전 방향전파 과 후 방향전파를 했고, 예측 값과 derivative를 계산함으로써, 모든 M 트레이닝 샘플에 대해서 말이죠, for loop 사용 없이 말입니다. 그러면, 기울기 강하 업데이트는, w는 w 빼기 러닝속도 곱하기 dw dw는 위에서 계산했고요, b는 b 빼기 러닝속도 곱하기 db로 업데이트 됩니다.

적어도 한번의 기울기 강하의 iteration을 for loop 없이 도입할 수 있다는 것은 꽤 멋진 것 같습니다.

Broadcasting example

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes	
Carb	56.0	0.0	4.4	68.0	
Protein	1.2	104.0	52.0	8.0	
Fat	1.8	135.0	99.0	0.9	
	59 cal	$\frac{56}{59} \approx 94.9\%$			

$A = \begin{bmatrix} & & & \\ & & & \\ & & & \end{bmatrix}$

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

$$\text{cal} = A.\text{sum}(\text{axis} = 0)$$

$$\text{percentage} = 100 * A / (\text{cal} \xrightarrow{\text{reshape}(1,4)} \uparrow(3,4) / (1,4))$$

칼럼의 합은 칼로리를 나타내고, 각각의 탄단지 항목을 칼로리로 나눈뒤 100을 곱하면 각각의 percentage가 된다. 어떻게 (3,4) 행렬을 (1,4)로 나눌까? 바로 이러한 Broadcasting때문이다.

Broadcasting example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \xrightarrow{100} = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{(m,n)} \xrightarrow{(2,3)} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix}_{(1,n)} \xrightarrow{(1,n) \rightsquigarrow (m,n)} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix} \xrightarrow{(2,3)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}_{(m,n)} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix}_{(m,1)} \xrightarrow{(m,n)} = \begin{bmatrix} 101 & 102 & 103 \\ 204 & 205 & 206 \end{bmatrix} \xleftarrow{(m,n)}$$