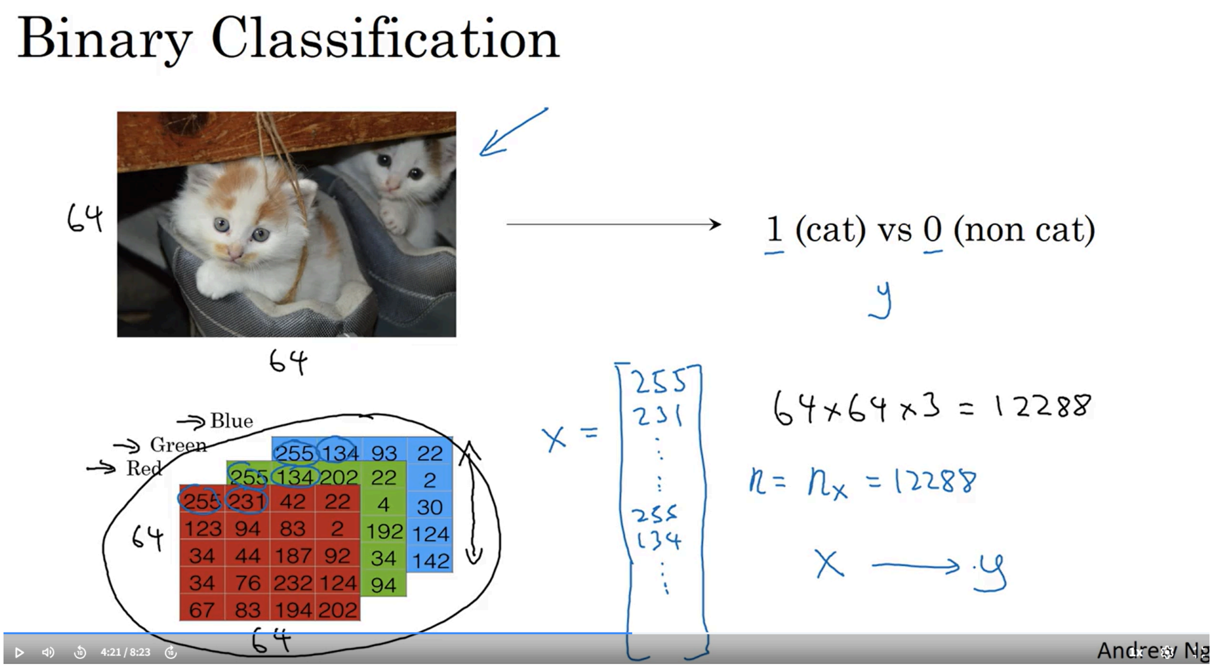
Logistic regression as a Neural Network

Binary Classification



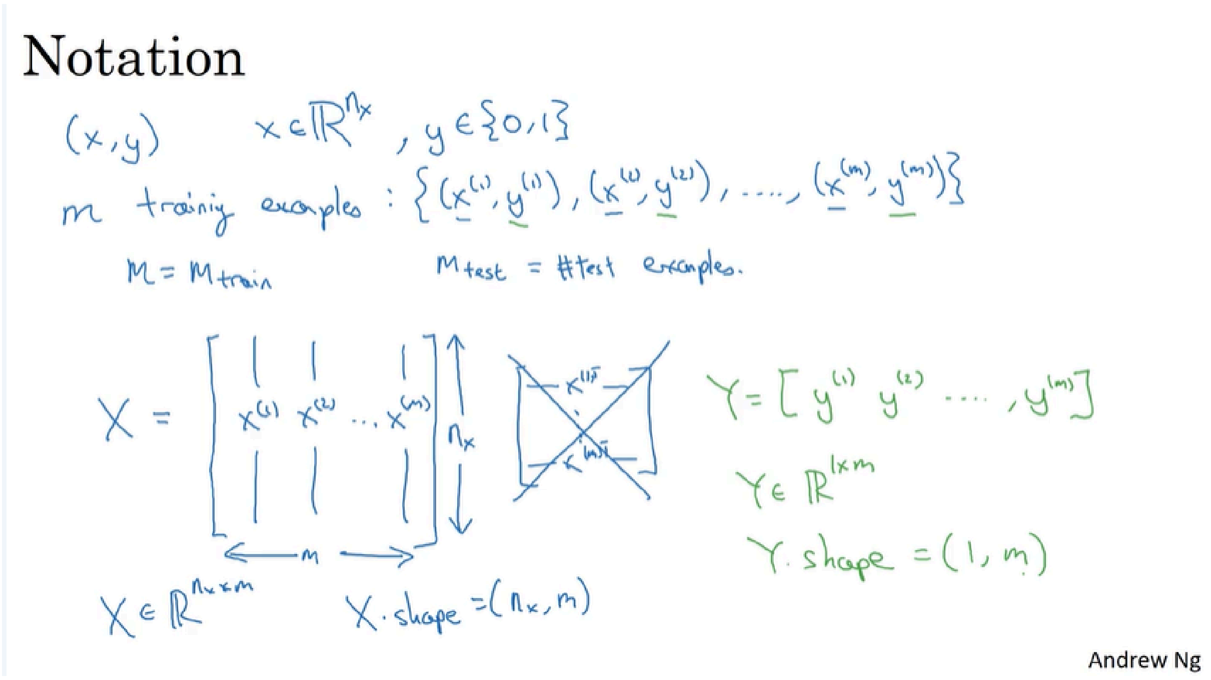
Logistic regression is an algorithm for binary classification.

To store an image your computer stores three separate matrices corresponding to the red, green, and blue color channels of this image.

So if your input image is 64 pixels by 64 pixels, then you would have 3 x (64 × 64) matrices corresponding to the red, green and blue pixel intensity values for your images.

Flatten it into a vector called x, with n_x = 3 ∗ 64 ∗ 64 = 12288

So in binary classification, our goal is to learn a classifier that can input an image represented by this feature vector x. And predict whether the corresponding label y is 1 or 0, that is, whether this is a cat image or a non-cat image.



Notation

$(x, y) \quad x \in \mathbb{R}^{n_x}, \; y \in \{0, 1\}$

m training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$M = M_{train} \qquad M_{test} = \#\text{test examples.}$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \; n_x \qquad \cancel{\begin{bmatrix} — x^{(1)T} — \\ \vdots \\ — x^{(m)T} — \end{bmatrix}}$$

$\xleftarrow{\quad m \quad}$

$X \in \mathbb{R}^{n_x \times m} \qquad X.\text{shape} = (n_x, m)$

$Y = [y^{(1)} \; y^{(2)} \; \ldots, y^{(m)}]$

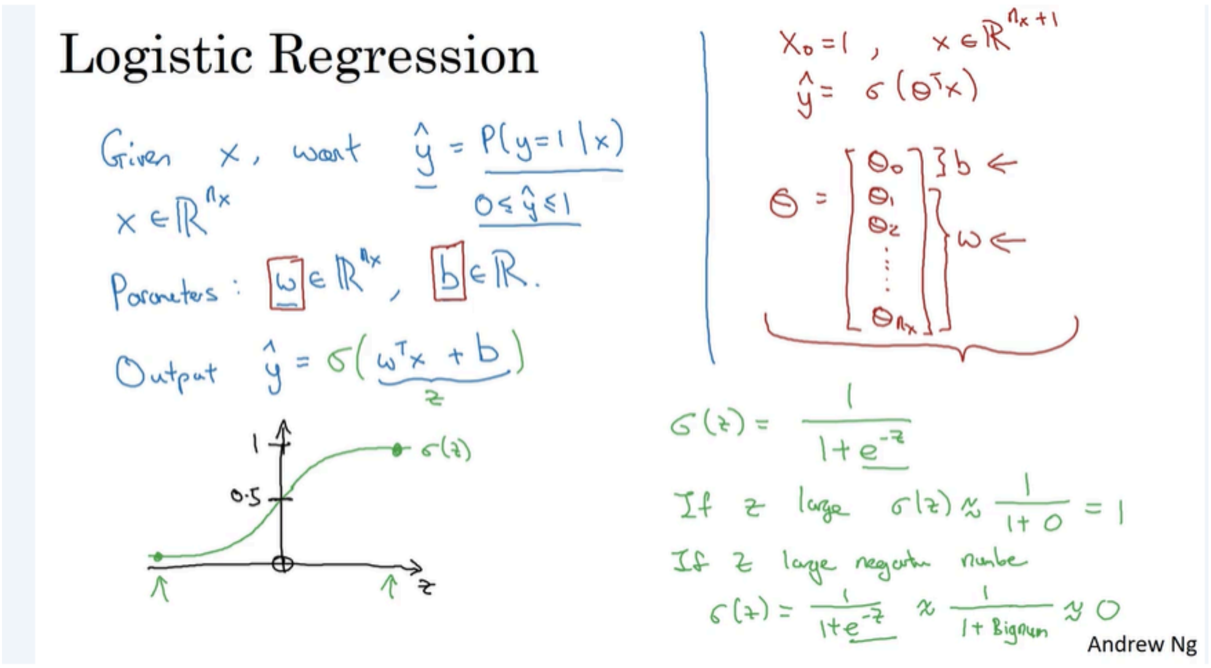
$Y \in \mathbb{R}^{1 \times m}$

$Y.\text{shape} = (1, m)$

Andrew Ng

$x^{(i)}$ is the i'th example (a vector) of the dataset with m examples. And we define X and Y like the image

Logistic Regression



Logistic Regression

Given x, want $\hat{y} = P(y=1 \mid x)$

$x \in \mathbb{R}^{n_x} \qquad 0 \leq \hat{y} \leq 1$

Parameters : $\boxed{w} \in \mathbb{R}^{n_x}, \; \boxed{b} \in \mathbb{R}.$

Output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$

1
0.5
$\sigma(z)$
z

$X_0 = 1, \quad x \in \mathbb{R}^{n_x + 1}$

$\hat{y} = \sigma(\theta^T x)$

$$\Theta = \underbrace{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix}}$$
$\} b \leftarrow \quad \} w \leftarrow$

$\sigma(z) = \dfrac{1}{1 + e^{-z}}$

If z large $\sigma(z) \approx \dfrac{1}{1+0} = 1$

If z large negative number

$\sigma(z) = \dfrac{1}{1+e^{-z}} \approx \dfrac{1}{1 + \text{Bignum}} \approx 0$

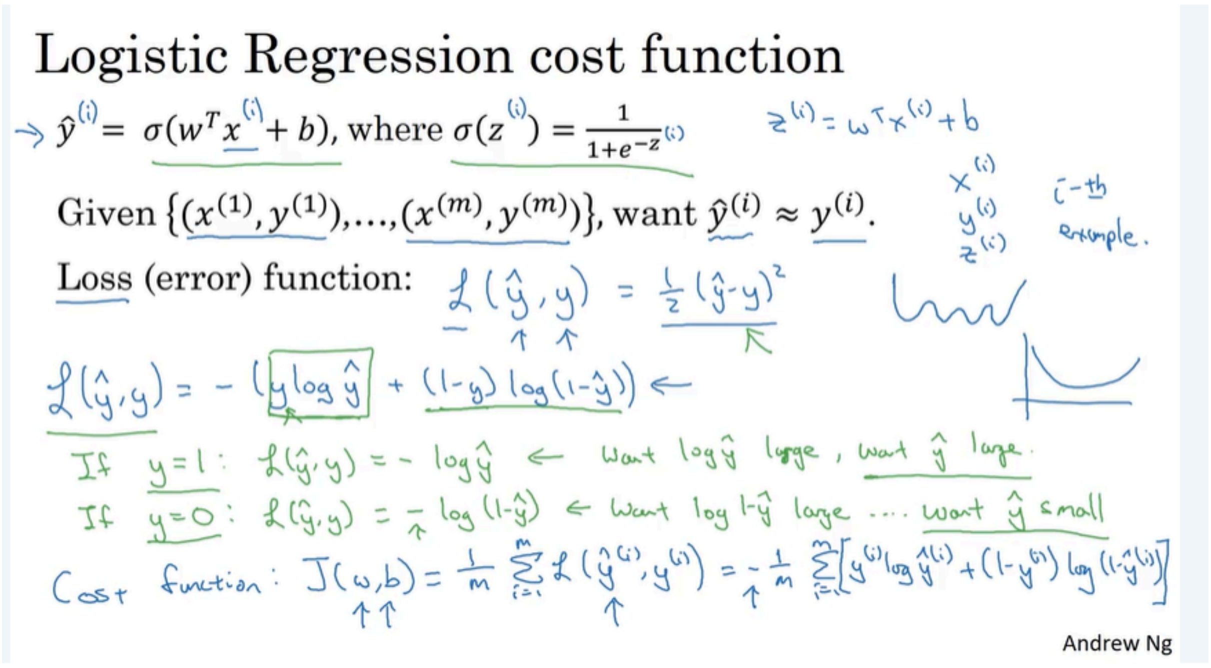
Andrew Ng

So when you implement logistic regression, your job is to try to learn parameters W and B so that Y hat becomes a good estimate of the chance of Y being equal to one.

And so, in this class, we will not use any of this notational convention (θ) that I just wrote in red. If you've not seen this notation before in other courses, don't worry about it.

It's just that for those of you that have seen this notation I wanted to mention explicitly that we're not using that notation in this course

Logistic Regression Cost Function



Logistic Regression cost function

$\rightarrow \hat{y}^{(i)} = \sigma(w^T x^{(i)} + b)$, where $\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}$ $\quad z^{(i)} = w^T x^{(i)} + b$

Given $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$.

$x^{(i)}$ $y^{(i)}$ $z^{(i)}$ i-th example.

Loss (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

$\mathcal{L}(\hat{y}, y) = -(\boxed{y \log \hat{y}} + (1-y)\log(1-\hat{y})) \leftarrow$

If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y} \leftarrow$ Want $\log \hat{y}$ large, want $\hat{y}$ large.

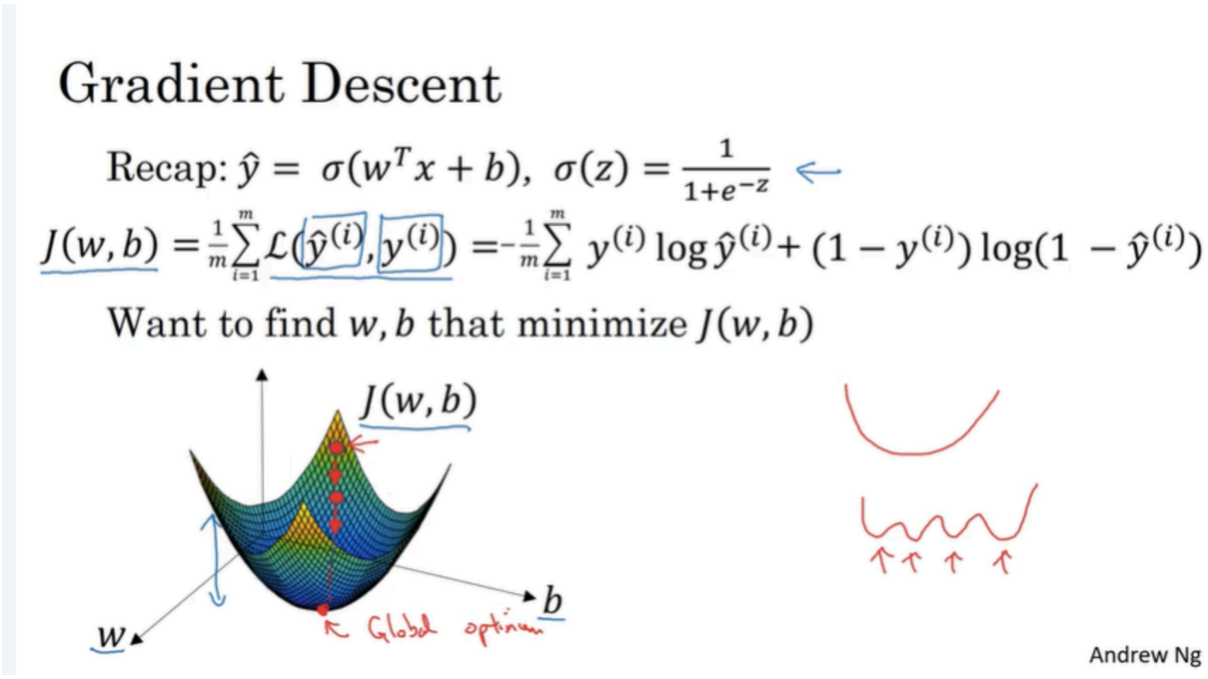
If $y=0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y}) \leftarrow$ Want $\log 1-\hat{y}$ large .... want $\hat{y}$ small

Cost function: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\hat{y}^{(i)} + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$

Andrew Ng

The cost function is the cost of your parameters, so in training your logistic regression model, we're going to try to find parameters W and B. That minimize the overall cost function J.

Gradient Descent



Gradient Descent

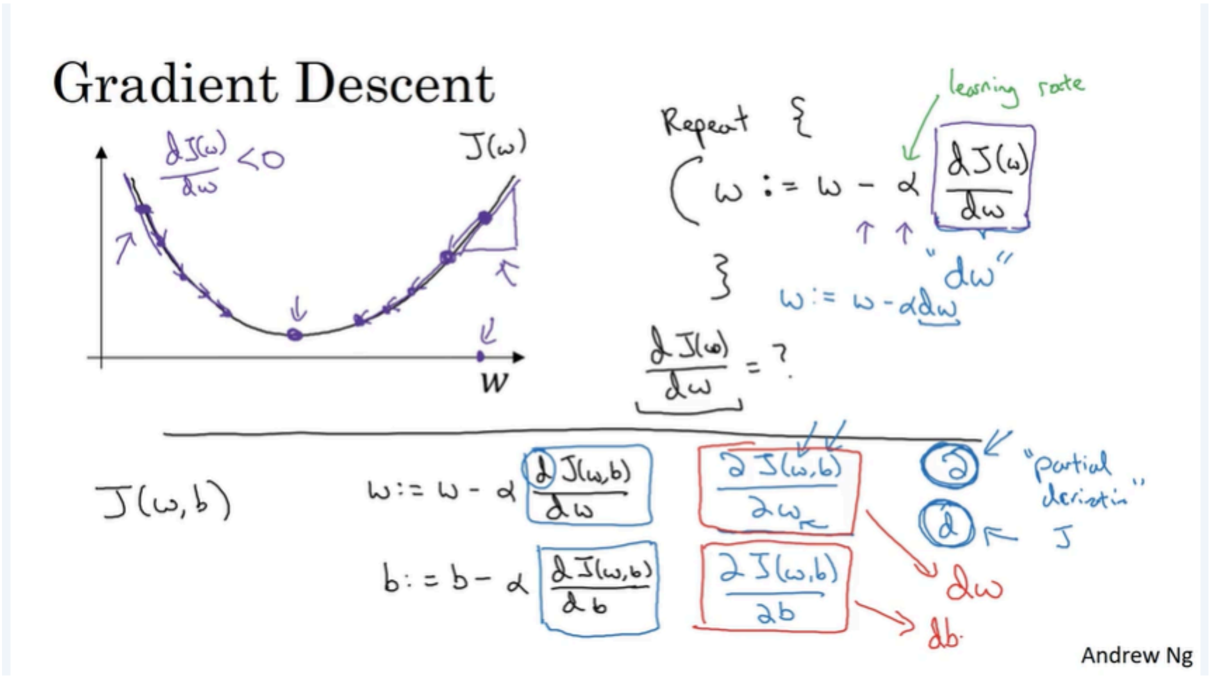
Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

$$\underline{J(w,b)} = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\boxed{\hat{y}^{(i)}}, \boxed{y^{(i)}}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$$

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

The loss function measures how well your algorithms outputs

The full formula is expanded out on the right. So the cost function measures how well your parameters w and b are doing on the training set



Gradient Descent

$$\text{Repeat } \{ \quad w := w - \alpha \boxed{\frac{dJ(w)}{dw}} \quad \}$$

$$w := w - \alpha \, dw$$

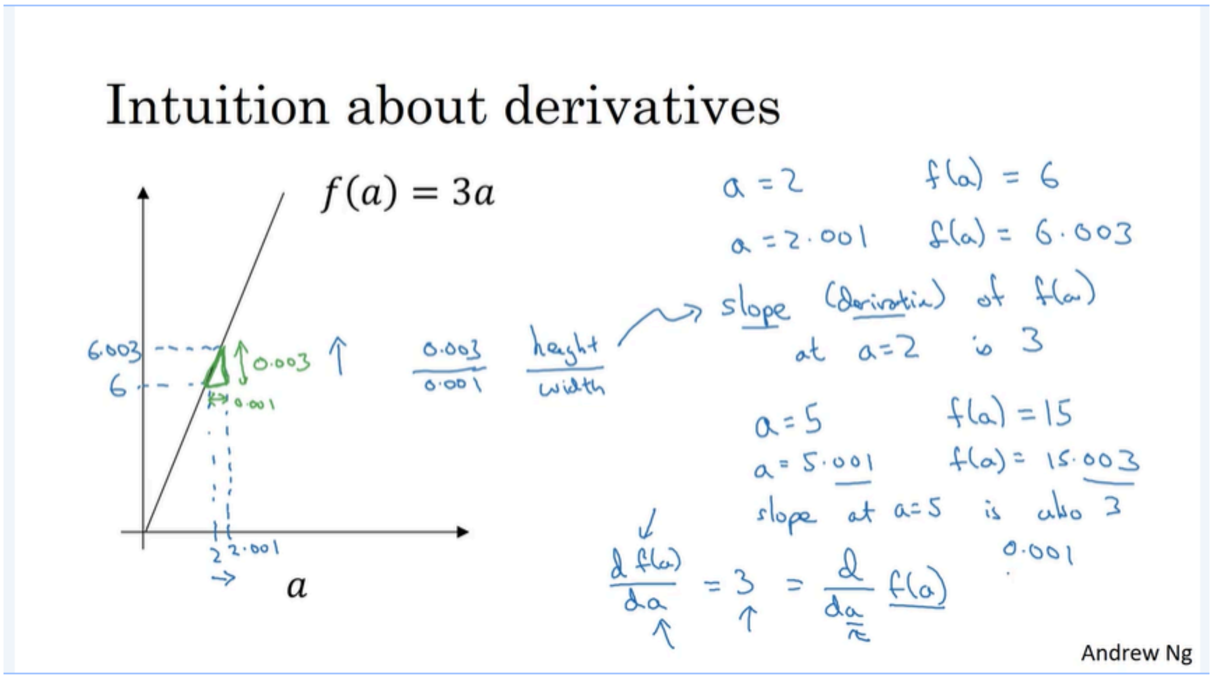
$$\underbrace{\frac{dJ(w)}{dw}}= ?$$

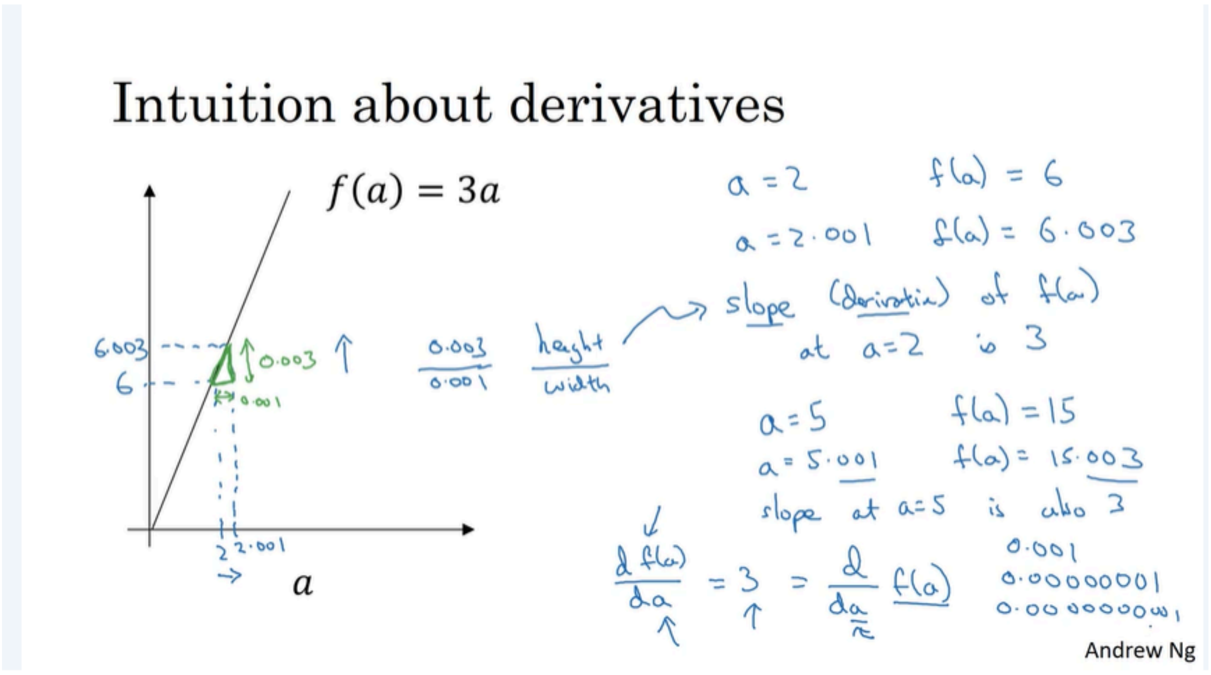
$$J(w,b) \qquad w := w - \alpha \boxed{\frac{dJ(w,b)}{dw}} \quad \boxed{\frac{\partial J(w,b)}{\partial w}} \rightarrow dw$$

$$b := b - \alpha \boxed{\frac{dJ(w,b)}{db}} \quad \boxed{\frac{\partial J(w,b)}{\partial b}} \rightarrow db$$

So you're at this point on the cost function J of w. Remember that the definition of a derivative is the slope of a function at the point. So the slope of the function is really, the height divided by the width right of the lower triangle.

Derivatives


Intuition about derivatives
$f(a) = 3a$
$a = 2$ $f(a) = 6$
$a = 2.001$ $f(a) = 6.003$
slope (derivative) of $f(a)$ at $a=2$ is 3
$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$
6.003
6
0.003
0.001
2 2.001
a
$a = 5$ $f(a) = 15$
$a = 5.001$ $f(a) = 15.003$
slope at $a=5$ is also 3
0.001
$\frac{d f(a)}{da} = 3 = \frac{d}{da} f(a)$
Andrew Ng


Intuition about derivatives
$f(a) = 3a$
$a = 2$ $f(a) = 6$
$a = 2.001$ $f(a) = 6.003$
slope (derivative) of $f(a)$ at $a=2$ is 3
$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$
6.003
6
0.003
0.001
2 2.001
a
$a = 5$ $f(a) = 15$
$a = 5.001$ $f(a) = 15.003$
slope at $a=5$ is also 3
0.001
0.00000001
0.000000001
$\frac{d f(a)}{da} = 3 = \frac{d}{da} f(a)$
Andrew Ng

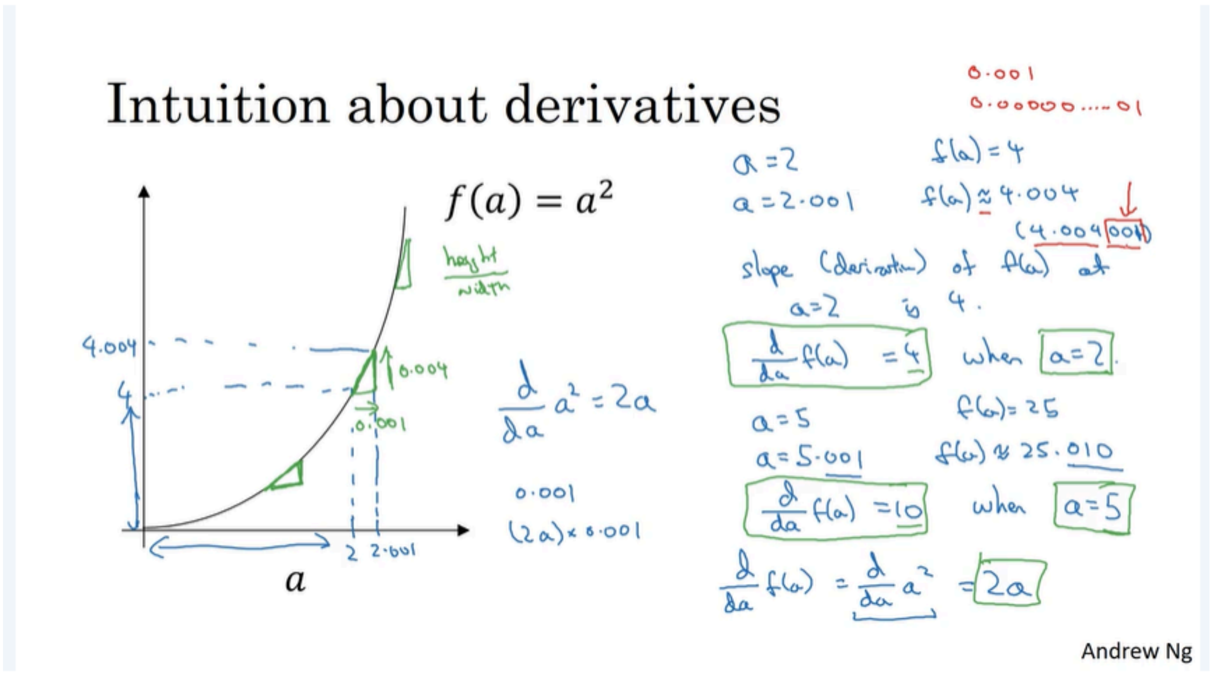
But all this equation means is that, if I nudge a to the right a little bit, I expect f(a) to go up by three times as much as I nudged the value of little a

Derivatives are defined with an even smaller value of how much you nudge (increase a really small number) a to the right. So, it's not 0.001. It's not 0.000001. It's not 0.00000000 and so on 1.

It's even smaller than that, and the formal definition of derivative says, whenever you nudge a to the right by an infinitesimal amount, basically an infinitely tiny, tiny amount. If you do that, this f(a) go up three times as much as whatever was the tiny, tiny, tiny amount that you nudged a to the right.

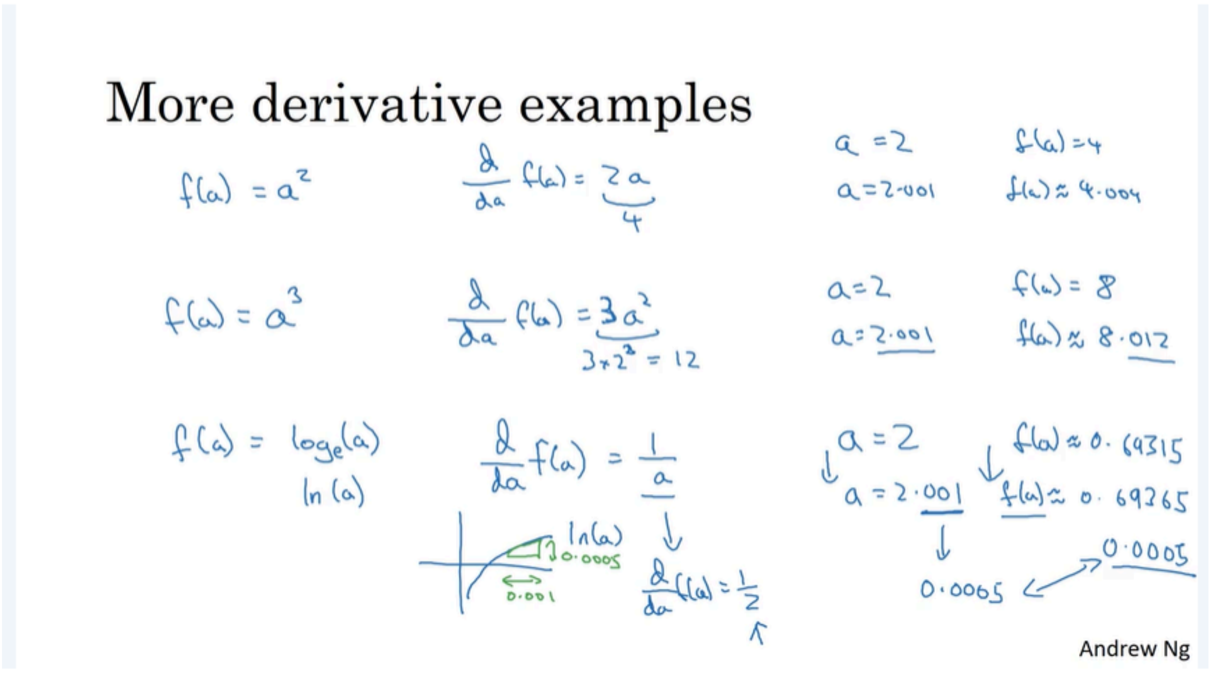
So, that's actually the formal definition of a derivative. But for the purposes of our intuitive understanding, which I'll talk about nudging a to the right by this small amount 0.001. Even if it's 0.001 isn't exactly tiny, tiny infinitesimal.

More Derivative Examples



Intuition about derivatives
$f(a) = a^2$
height / width
4.004
4
0.004
0.001
2
2.001
a
$\frac{d}{da} a^2 = 2a$
0.001
(2a) × 0.001
0.001
0.00000....01
a = 2   f(a) = 4
a = 2.001   f(a) ≈ 4.004
(4.004001)
slope (derivative) of f(a) at a = 2 is 4.
$\boxed{\frac{d}{da} f(a) = 4}$ when $\boxed{a = 2}$
a = 5   f(a) = 25
a = 5.001   f(a) ≈ 25.010
$\boxed{\frac{d}{da} f(a) = 10}$ when $\boxed{a = 5}$
$\frac{d}{da} f(a) = \frac{d}{da} a^2 = \boxed{2a}$
Andrew Ng

So one way to see why did derivatives is different at different points is that if you draw that little triangle right at different locations on this, you'll see that the ratio of the height of the triangle over the width of the triangle is very different at different points on the curve.



More derivative examples

$f(a) = a^2$ $\quad \frac{d}{da} f(a) = \underbrace{2a}_{4}$ $\quad a = 2 \quad f(a) = 4$ $\quad a = 2.001 \quad f(a) \approx 4.004$

$f(a) = a^3$ $\quad \frac{d}{da} f(a) = 3a^2$ $\quad 3 \times 2^2 = 12$ $\quad a = 2 \quad f(a) = 8$ $\quad a = 2.001 \quad f(a) \approx 8.012$

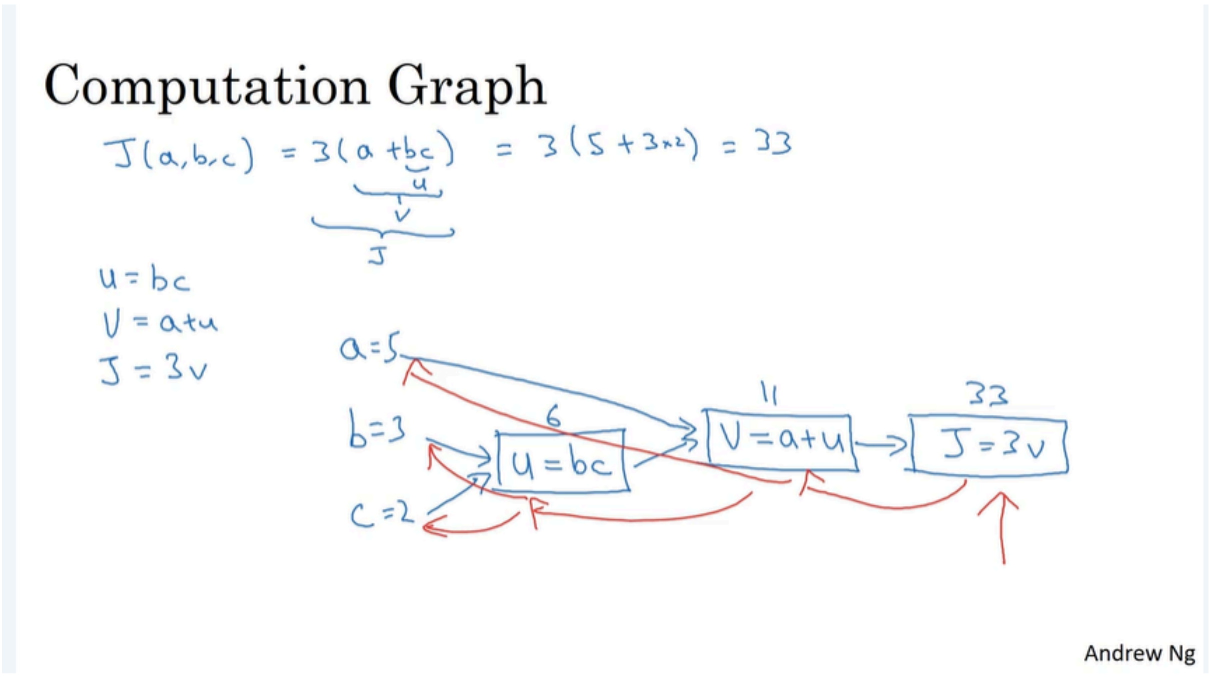
$f(a) = \log_e(a)$ $\quad \ln(a)$ $\quad \frac{d}{da} f(a) = \frac{1}{a}$ $\quad \frac{d}{da} f(a) = \frac{1}{2}$ $\quad a = 2 \quad f(a) \approx 0.69315$ $\quad a = 2.001 \quad f(a) \approx 0.69365$ $\quad 0.0005 \quad 0.0005$

ln(a)
0.0005
0.001
Andrew Ng

There are just two take home messages from this video.

- First is that the derivative of the function just means the slope of a function and the slope of a function can be different at different points on the function. In our first example where f(a) = 3a those a straight line. The derivative was the same everywhere, it was three everywhere. For other functions like f(a) = a^2 or f(a) = log(a), the slope of the line varies. So, the slope or the derivative can be different at different points on the curve.
- Second takeaway is that if you want to look up the derivative of a function, you can flip open your calculus textbook or look up Wikipedia and often get a formula for the slope of these functions at different points

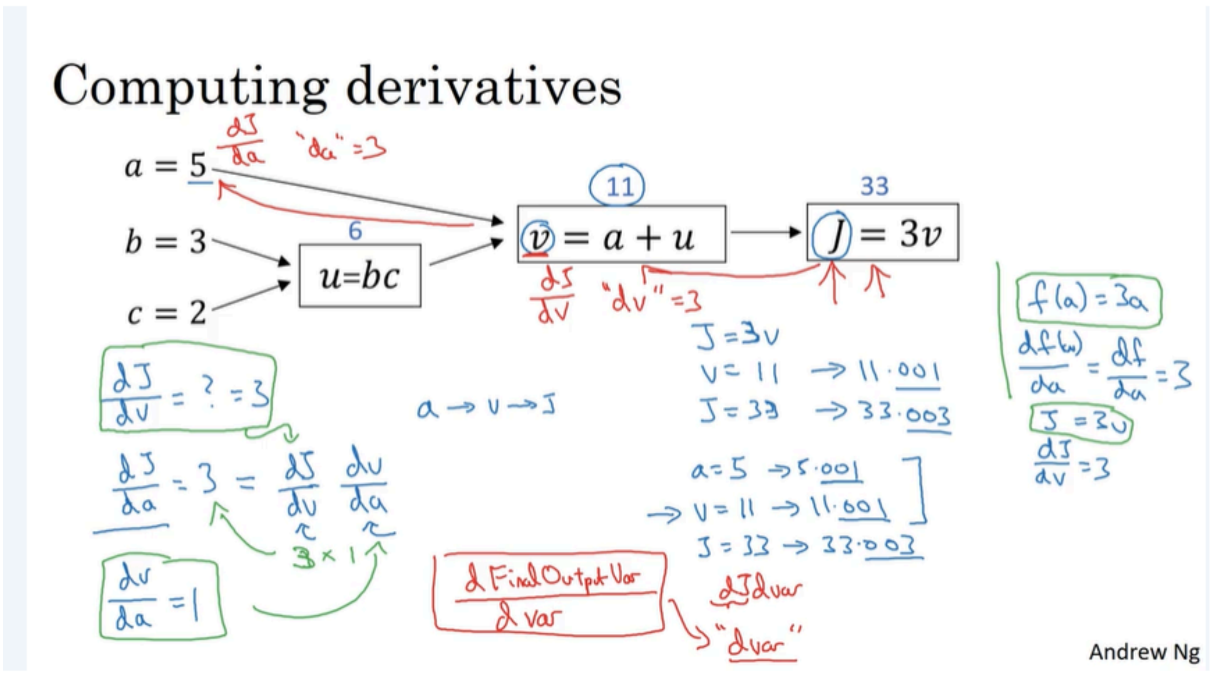
Computation Graph



The computation graph organizes a computation with this blue arrow, left-to-right computation. Let's refer to the next video how you can do the backward red arrow right-to-left computation of the derivatives.

One step of Backward propagation on a computation graph yields derivative of final output variable.

Derivatives with a Computation Graph



If we were to take this value of v and change it a little bit, how would the value of J change? Well, J is defined as 3 times v. And right now, v = 11. So if we're to bump up v by a little bit to 11.001,

then J, which is 3v, so currently 33, will get bumped up to 33.003. So here, we've increased v by 0.001. And the net result of that is that J goes up 3 times as much. So the derivative of J with respect to v is equal to 3.

Because the increase in J is 3 times the increase in v. And in fact, this is very analogous to the example we had in the previous video, where we had f(a) = 3a. And so we then derived that df/da, which with slightly simplified, a slightly sloppy notation, you can write as df/da = 3. So instead, here we have J = 3v, and so dJ/dv = 3. With here, J playing the role of f, and v playing the role of a in this previous example that we had from an earlier video.

So indeed, terminology of backpropagation, what we're seeing is that if you want to compute the derivative of this final output variable, which usually is a variable you care most about, with respect to v, then we've done one step of backpropagation. So we call it one step backwards in this graph. Now let's look at another example. What is dJ/da? In other words, if we bump up the value of a, how does that affect the value of J?

Well, let's go through the example, where now a = 5. So let's bump it up to 5.001. The net impact of that is that v, which was a + u, so that was previously 11. This would get increased to 11.001. And then we've already seen as above that J now gets bumped up to 33.003. So what we're seeing is that if you increase a by 0.001, J increases by 0.003. And by increase a, I mean, you have to take this value of 5 and just plug in a new value. Then the change to a will propagate to the right of the computation graph so that J ends up being 33.003. And so the increase to J is 3 times the increase to a. So that means this derivative is equal to 3. And one way to break this down is to say that if you change a, then that will change v.

And through changing v, that would change J. And so the next change to the value of J when you bump up the value, when you nudge the value of a up a little bit, is that

First, by changing a, you end up increasing v. Well, how much does v increase? It is increased by an amount that's determined by dv/da. And then the change in v will cause the value of J to also increase. So in calculus, this is actually called the chain rule that if a affects v, affects J, then the amounts that J changes when you nudge a is the product of how much v changes when you nudge a times how much J changes when you nudge v.

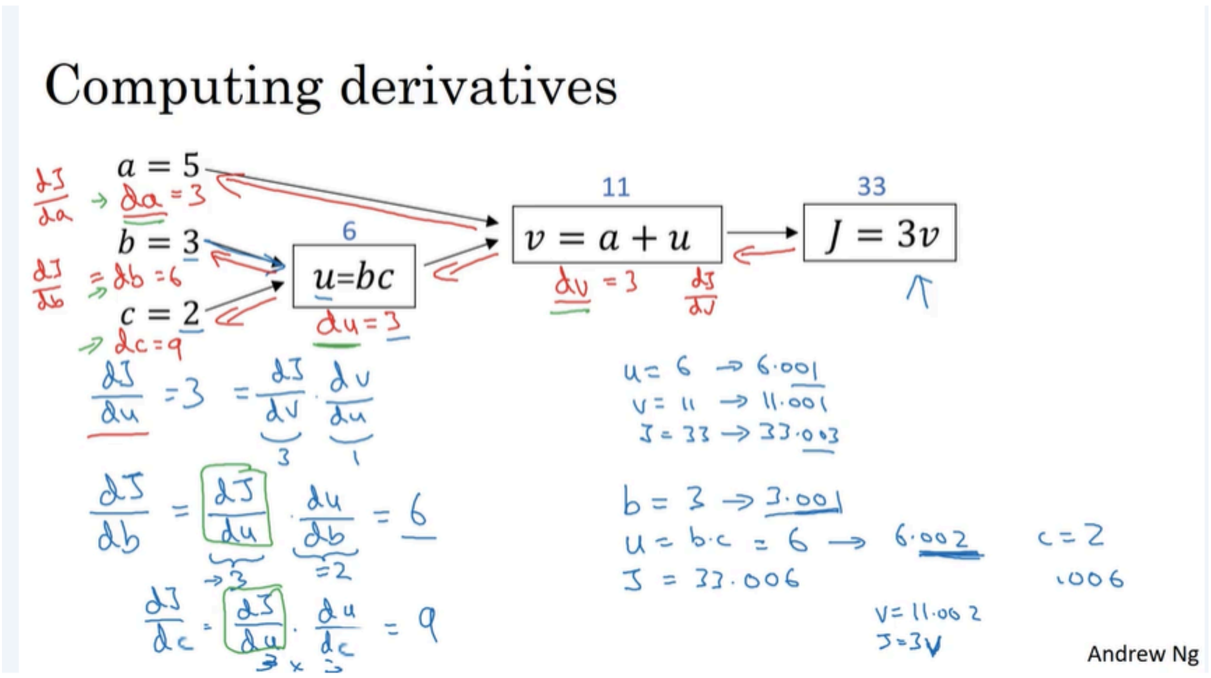
So in calculus, again, this is called the chain rule. And what we saw from this calculation is that if you increase a by 0.001, v changes by the same amount. So dv/da = 1. So in fact, if you plug in what we have wrapped up previously, dv/dJ = 3 and dv/da = 1. So the product of these 3 times 1, that actually gives you the correct value that dJ/da = 3. So this little illustration shows hows by having computed dJ/dv, that is, derivative with respect to this variable, it can then help you to compute dJ/da. And so that's another step of this backward calculation.

I just want to introduce one more new notational convention. Which is that when you're witting codes to implement backpropagation, there will usually be some final output variable that you really care about. So a final output variable that you really care about or that you want to optimize. And in this case, this final output variable is J. It's really the last node in your computation graph. And so a lot of computations will be trying to compute the derivative of that final output variable. So d of this final output variable with respect to some other variable. Then we just call that dvar.

So a lot of the computations you have will be to compute the derivative of the final output variable, J in this case, with various intermediate variables, such as a, b, c, u or v. And when you implement this in software, what do you call this variable name? One thing you could do is in Python, you could give us a very long variable name like dFinalOurputVar/dvar. But that's a very long variable name. You could call this, I guess, dJdvar.

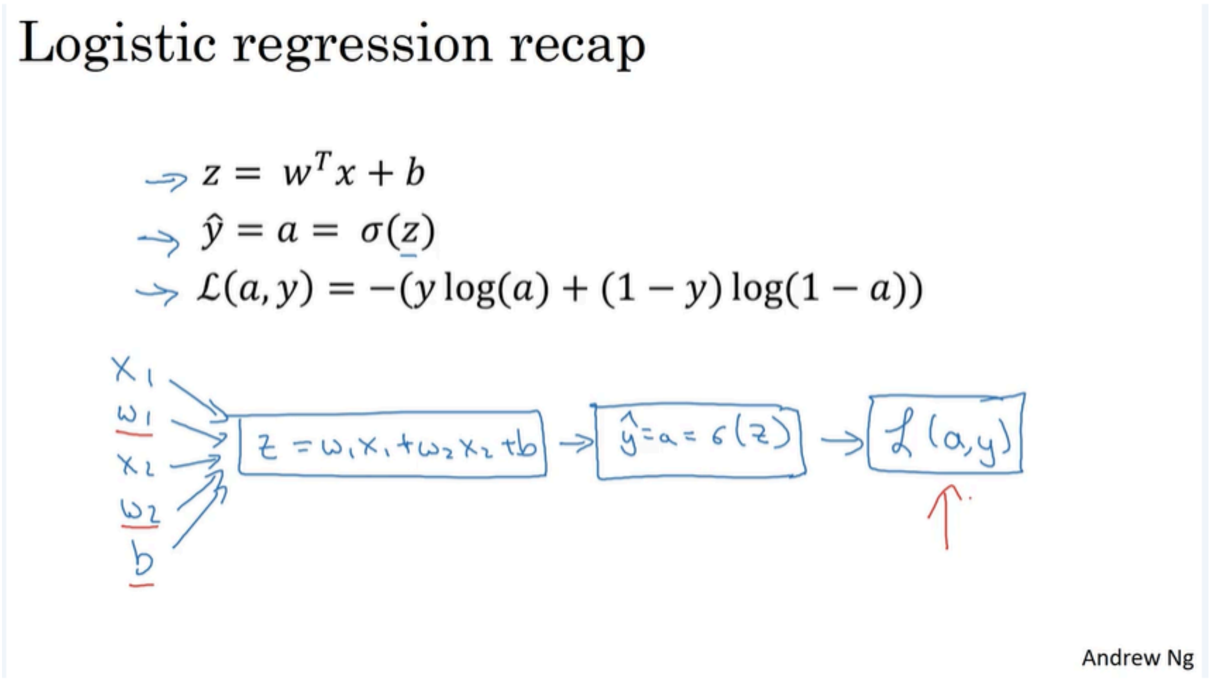
But because you're always taking derivatives with respect to dJ, with respect to this final output variable, I'm going to introduce a new notation. Where, in code, when you're computing this thing in the code you write, we're just going to use the variable name dvar in order to represent that quantity. So dvar in a code you write will represent the derivative of the final output variable you care about such as J. Well, sometimes, the last I with respect to the various intermediate quantities you're computing in your code.

So this thing here in your code, you use dv to denote this value. So dv would be equal to 3. And your code, you represent this as da, which is we also figured out to be equal to 3. So we've done backpropagation partially through this computation graph. Let's go through the rest of this example on the next slide.



The key takeaway from this video, from this example, is that when computing derivatives and computing all of these derivatives, the most efficient way to do so is through a right to left computation following the direction of the red arrows. And in particular, we'll first compute the derivative with respect to v. And then that becomes useful for computing the derivative with respect to a and the derivative with respect to u. And then the derivative with respect to u, for example, this term over here (dJ/du in green rectangle) and this term over here. Those in turn become useful for computing the derivative with respect to b and the derivative with respect to c.

Logistic Regression Gradient Descent

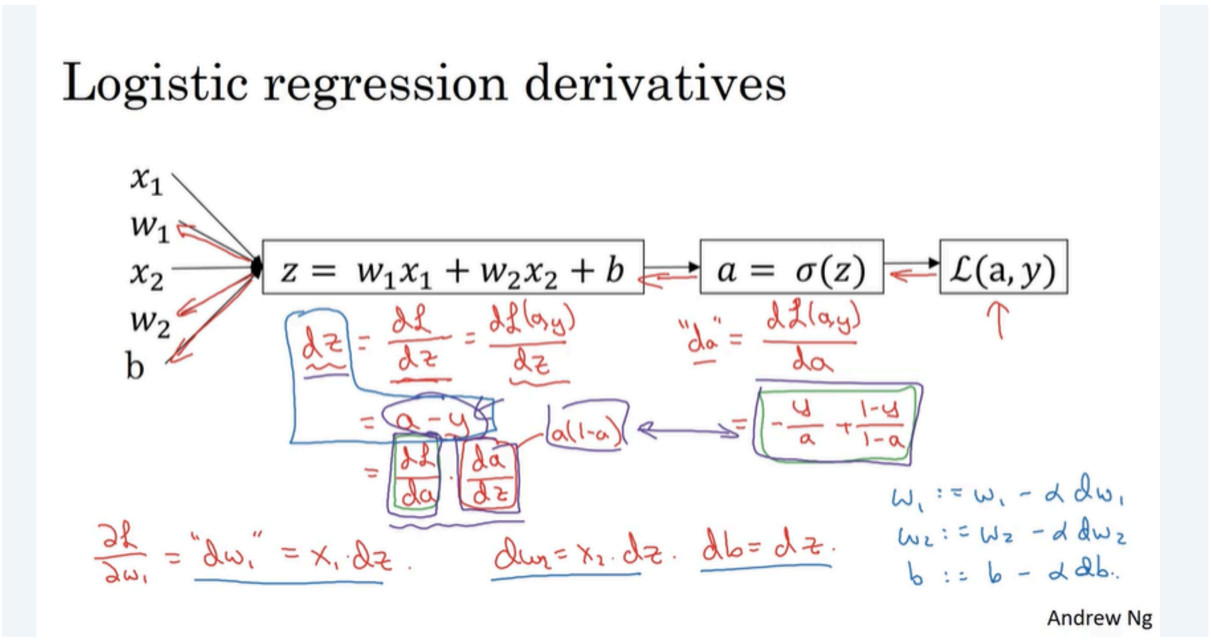


Logistic regression recap

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

$$\begin{tikzcd} x_1, w_1, x_2, w_2, b \arrow[r] & \boxed{z = w_1x_1 + w_2x_2 + b} \arrow[r] & \boxed{\hat{y} = a = \sigma(z)} \arrow[r] & \boxed{\mathcal{L}(a,y)} \end{tikzcd}$$

Andrew Ng



Logistic regression derivatives

$$\begin{tikzcd} x_1, w_1, x_2, w_2, b \arrow[r] & \boxed{z = w_1x_1 + w_2x_2 + b} \arrow[r] & \boxed{a = \sigma(z)} \arrow[r] & \boxed{\mathcal{L}(a,y)} \end{tikzcd}$$

$$\boxed{dz} = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz} \qquad \text{"}da\text{"} = \frac{d\mathcal{L}(a,y)}{da}$$

$$= a - y \qquad a(1-a) \longleftrightarrow = \boxed{-\frac{y}{a} + \frac{1-y}{1-a}}$$

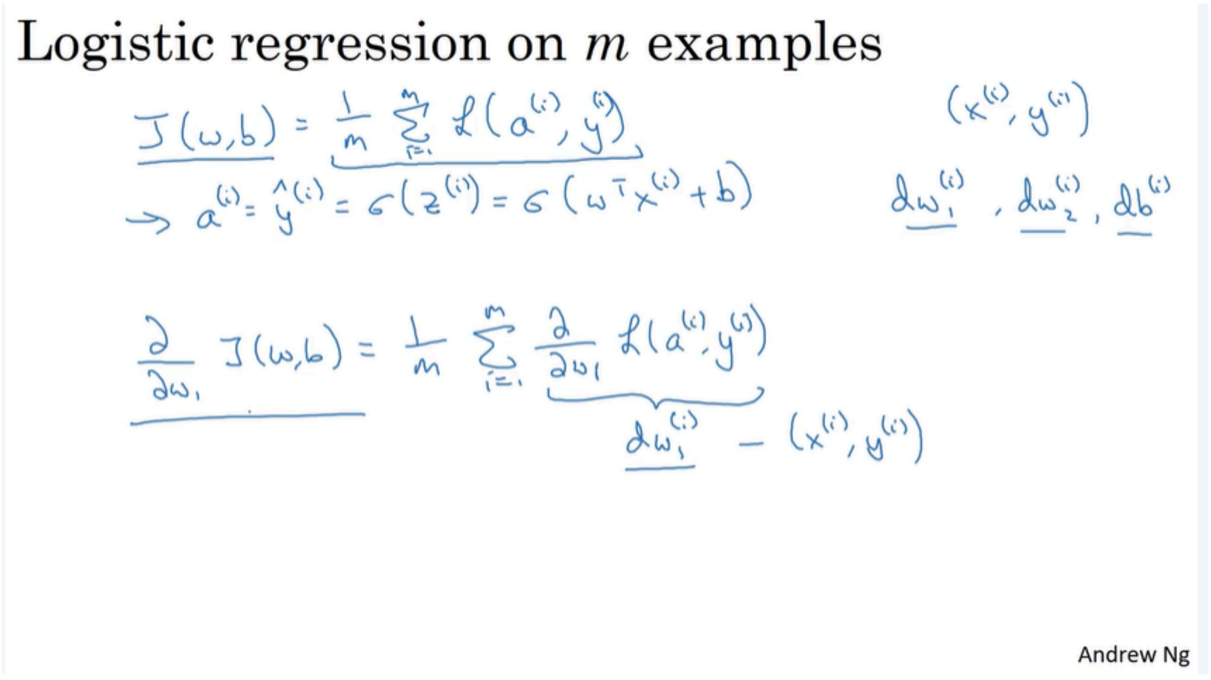
$$= \boxed{\frac{d\mathcal{L}}{da}} \cdot \boxed{\frac{da}{dz}}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \text{"}dw_1\text{"} = x_1 \cdot dz. \qquad dw_2 = x_2 \cdot dz. \qquad db = dz.$$

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db.$$

Andrew Ng

Gradient Descent on m Examples



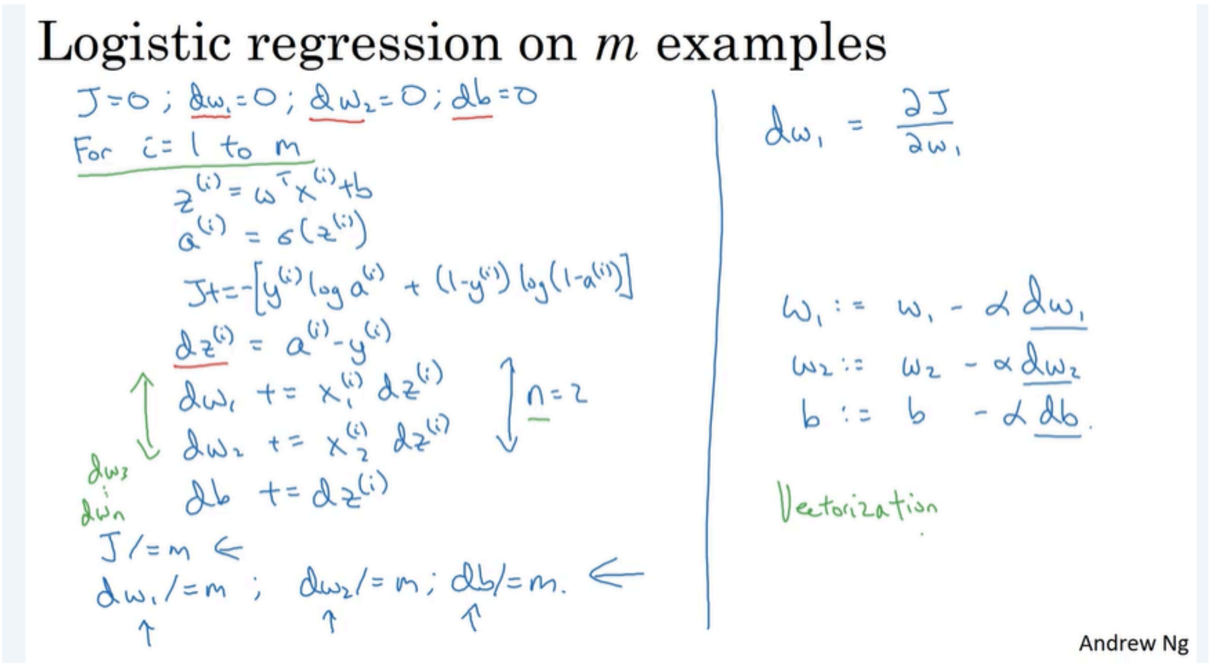
Logistic regression on m examples

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

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b) \qquad \underline{dw_1^{(i)}},\ \underline{dw_2^{(i)}},\ \underline{db^{(i)}}$$

$$\underline{\frac{\partial}{\partial w_1} J(w,b)} = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1}\mathcal{L}(a^{(i)}, y^{(i)})}_{\underline{dw_1^{(i)}}\ -\ (x^{(i)},\, y^{(i)})}$$

Andrew Ng



Logistic regression on m examples

```
$J = 0$; $dw_1 = 0$; $dw_2 = 0$; $db = 0$
For $i = 1$ to $m$
        $z^{(i)} = w^T x^{(i)} + b$
        $a^{(i)} = \sigma(z^{(i)})$
        $J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$
        $dz^{(i)} = a^{(i)} - y^{(i)}$
    ↑   $dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$     ↑ $n = 2$
    ↓   $dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$     ↓
 $dw_3$
   ⋮
 $dw_n$  $db \mathrel{+}= dz^{(i)}$
  $J /= m$ ←
  $dw_1 /= m$;  $dw_2 /= m$;  $db /= m$. ←
      ↑            ↑           ↑
```

$$dw_1 = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha\, \underline{dw_1}$$
$$w_2 := w_2 - \alpha\, \underline{dw_2}$$
$$b := b - \alpha\, \underline{db}.$$

Vectorization

Andrew Ng

This is learned in course Machine Learning Specialization

In the for loop depicted in the video, why is there only one *dw* variable (i.e. no i superscripts in the for loop)?

→ The value of *dw* in the code is cumulative.