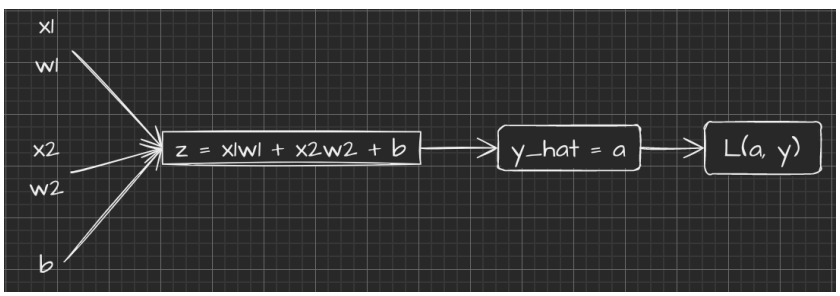
Logistic Regression Gradient Descent

Key takeaway: what do you need to implement?

Logistic Regression Recap:

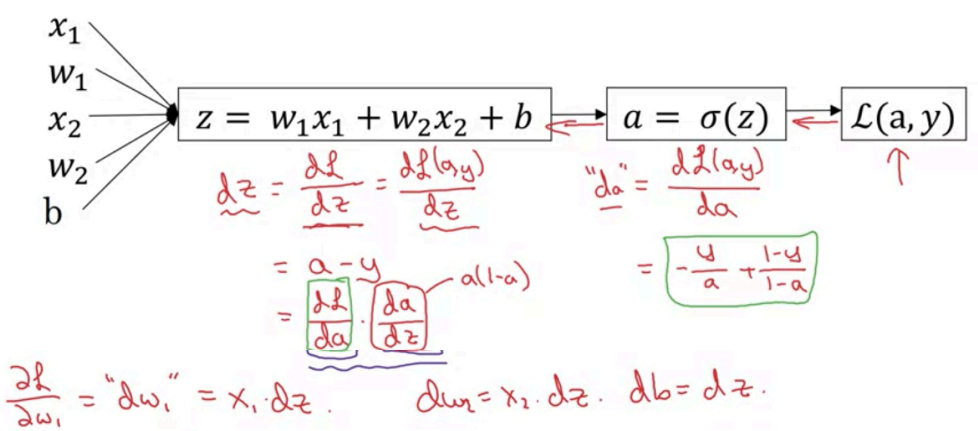
$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$

Let's say, for example, we have one training example with only 2 features x1 and x2, come with 2 parameters w1 and w2, and the bias parameter b. We have the following computation graph:



-> We want to modify the parameters w and b in order to reduce the loss.

Now, we're going to describe the backwards propagation step to compute the derivatives.



Example:

$$\frac{dl}{dw_1} = \frac{dl}{da} \cdot \frac{da}{dz} \cdot \frac{dz}{dw_1}$$
$$= \left(\frac{-y}{a} + \frac{1-y}{1-a}\right) \cdot (a(1 - a)) \cdot (x_1)$$
$$= (a - y)x_1$$

To compute Gradient Descent with respect to 1 training example that have, for example, 2 features:

1. Compute dz using the formula: dz = a - y
2. Compute dw_1, dw_2 using the formula: dw_1 = x_1dz, dw_2 = x_2dz
3. Compute db = dz
4. Perform the updates using these formulas:

$$
\begin{aligned}
w_1 &:= w_1 - \alpha \cdot dw_1 \\
w_2 &:= w_2 - \alpha \cdot dw_2 \\
b &:= b - \alpha \cdot db
\end{aligned}
$$

Logistic Regression on m training examples

The general form of the Cost function:

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

$$\text{where } a^{(i)} \text{ is the prediction on the } i^{th} \text{ training example}$$

$$\Rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

What we showed in the previous slides is for any single training example, how to compute the derivatives when you have just one training example.

It turns out that the derivative with respect to w_1 **of the overall cost function** is going to be the average of derivatives respect to w_1 of the individual lost term.

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

Previously, we have shown how to compute this term as dw_1(ith) on a single training example.

-> What we need to do is compute these derivatives as we shown on the previous training example and average them.

Wrap it up to an concrete algorithm to implement logistic regression with gradient descent:

Let's initialize J = 0, dw_1 = 0, dw_2 = 0, db = 0, m = number of training examples

What we'e going to do is using a for loop over the training set and compute the derivative with respect to each training example and then add them up.

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

$$For\ i = 1\ \ to\ \ m:$$

$$z^{(i)} = w^T x^{(i)} + b$$

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

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

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 += x_1^{(i)} \cdot dz^{(i)}$$

$$dw_2 += x_2^{(i)} \cdot dz^{(i)}$$

$$(assuming\ that\ you\ have\ just\ 2\ features)$$

$$db += dz^{(i)}$$

$$J/ = m,\ \ dw_1/ = m,\ \ dw_2/ = m,\ \ db/ = m$$

With all these calculations, we've just computed the derivatives of the cost function J with respect to each parameters w_1, w_2, b

Notice that dw_1 and dw_2 do not have a superscript i, because we're using them in this code as accumulators to sum over the entire training set. Whereas in contrast, dz_i here, this was dz with respect to just one single training example.

(Cái dw_1, dw_2 sẽ cộng dồn lại từ các training examples riêng lẻ rồi lấy trung bình còn cái dz chỉ là dz của một training example riêng lẻ)

Having finished all these calculations, to implement one step of gradient descent, you will implement:

$$\begin{aligned} w_1\ &:=\ w_1\ -\ \alpha \cdot dw_1 \\ w_2\ &:=\ w_2\ -\ \alpha \cdot dw_2 \\ b\ &:=\ b\ -\ \alpha \cdot db \end{aligned}$$

You have to **repeat everything** on this slide **multiple time** in order to take multiple step of gradient descent because this slide demonstrates only 1 step of gradient descent.

But it turns out there are two weaknesses with the calculation as we've implemented it here, which is that, to implement logistic regression this way, **you need to write two for loops**:

- The first for loop is this for loop over the m training examples
- The second for loop is a for loop over all the features

When you're implementing deep learning algorithms, you find that having **explicit for loops** in your code **makes your algorithm run less efficiency**.

So, in the deep learning era, we would move to a bigger and bigger datasets, and so being able to implement your algorithms without using explicit for loops is really important and will help you to scale to much bigger datasets. So, it turns out that there are a set of techniques called **vectorization** techniques that allow you to **get rid of these explicit for-loops** in your code. I think in the pre-deep learning era, that's before the rise of deep learning, vectorization was a nice to have, so you could sometimes do it to speed up your code and sometimes not. But in the deep learning era, vectorization, that is getting rid of for loops, like this and like this, has become really important, because we're more and more training on very large datasets, and so you really need your code to be very efficient.