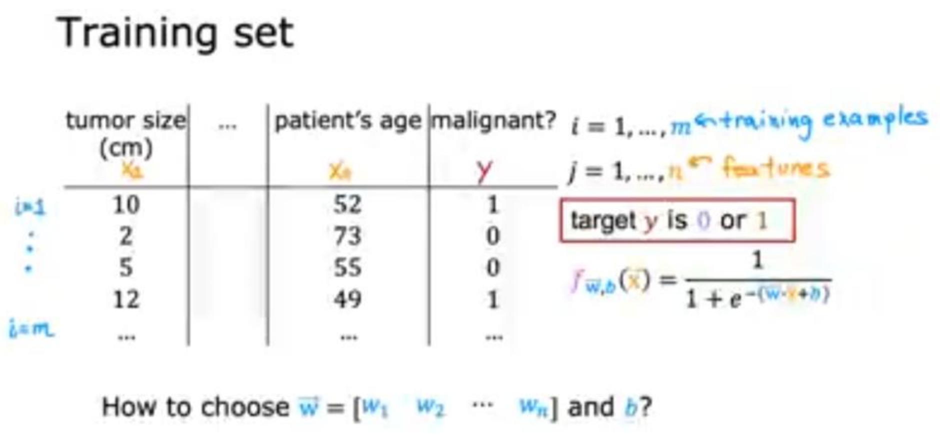
Fonction de coût pour la régression logistique

Remember that the cost function gives you a way to measure how well a specific set of parameters fits the training data. Thereby gives you a way to try to choose better parameters. In this video, we'll look at how the squared error cost function is not an ideal cost function for logistic regression. We'll take a look at a different cost function that can help us choose better parameters for logistic regression. Here's what the training set for our logistic regression model might look like.

Where here each row might correspond to patients that was paying a visit to the doctor and one dealt with some diagnosis. As before, we'll use m to denote the number of training examples. Each training example has one or more features, such as the tumor size, the patient's age, and so on for a total of n features. Let's call the features X_1 through X_n. Since this is a binary classification task, the target label y takes on only two values, either 0 or 1. Finally, the logistic regression model is defined by this equation. The question you want to answer is, given this training set, how can you choose parameters w and b?



Training set

	tumor size (cm) x_1	...	patient's age x_n	malignant? y
$i=1$	10		52	1
⋮	2		73	0
	5		55	0
	12		49	1
$i=m$	...		...	...

$i = 1, \ldots, m$ ← training examples

$j = 1, \ldots, n$ ← features

$\boxed{\text{target } y \text{ is } 0 \text{ or } 1}$

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

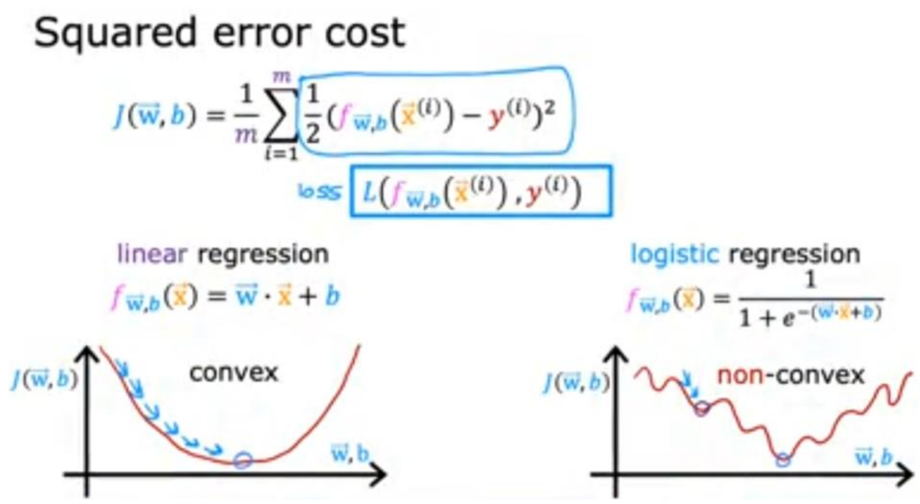
How to choose $\vec{w} = [w_1 \quad w_2 \quad \cdots \quad w_n]$ and b?

Recall for linear regression, this is the squared error cost function. The only thing I've changed is that I put the one half inside the summation instead of outside the summation. You might remember that in the case of linear regression, where f of x is the linear function, w dot x plus b. The cost function looks like this, is a convex function or a bowl shape or hammer shape. Gradient descent will look like this, where you take one step, one step, and so on to converge at the global minimum. Now you could try to use the same cost function for logistic regression.

But it turns out that if I were to write f of x equals 1 over 1 plus e to the negative wx plus b and plot the cost function using this value of f of x, then the cost will look like this. This becomes what's called a non-convex cost function is not convex. What this means is that if you were to try to use gradient descent. There are lots of local minima that you can get sucking. It turns out that for logistic regression, this squared error cost function is not a good choice. Instead, there will be a different cost function that can make the cost function convex again.

The gradient descent can be guaranteed to converge to the global minimum. The only thing I've changed is that I put the one half inside the summation instead of outside the summation. This will make the math you see later on this slide a little bit simpler. In order to build a new cost function, one that we'll use for logistic regression. I'm going to change a little bit the definition of the cost function J of w and b. In particular, if you look inside this summation, let's call this term inside the loss on a single training example.

I'm going to denote the loss via this capital L and as a function of the prediction of the learning algorithm, f of x as well as of the true label y. The loss given the predictor f of x and the true label y is equal in this case to 1.5 of the squared difference. We'll see shortly that by choosing a different form for this loss function, will be able to keep the overall cost function, which is 1 over n times the sum of these loss functions to be a convex function.



Squared error cost

$$J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} \boxed{\frac{1}{2}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2}$$

loss $\boxed{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)}$

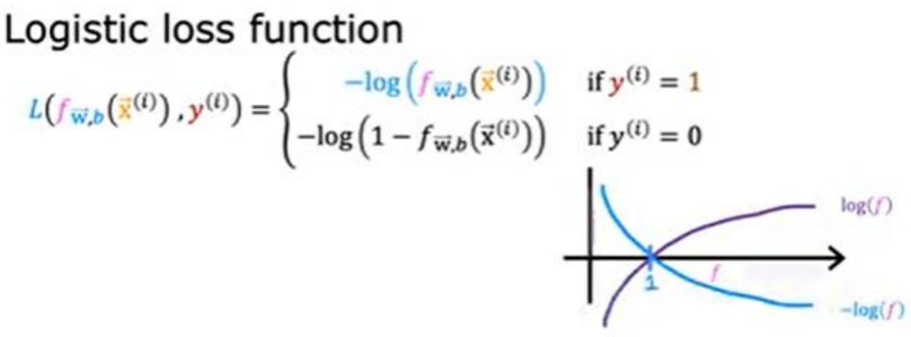
linear regression $f_{\vec{w},b}(\vec{x}) = \vec{w}\cdot\vec{x} + b$

logistic regression $f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\cdot\vec{x}+b)}}$

Now, the loss function inputs f of x and the true label y and tells us how well we're doing on that example. I'm going to just write down here at the definition of the loss function we'll use for logistic regression. If the label y is equal to 1, then the loss is negative log of f of x and if the label y is equal to 0, then the loss is negative log of 1 minus f of x.

Let's take a look at why this loss function hopefully makes sense. Let's first consider the case of y equals 1 and plot what this function looks like to gain some intuition about what this loss function is doing. Remember, the loss function measures how well you're doing on one training example and is by summing up the losses on all of the training examples that you then get, the cost function, which measures how well you're doing on the entire training set.

If you plot log of f, it looks like this curve here, where f here is on the horizontal axis. A plot of a negative of the log of f looks like this, where we just flip the curve along the horizontal axis. Notice that it intersects the horizontal axis at f equals 1 and continues downward from there.



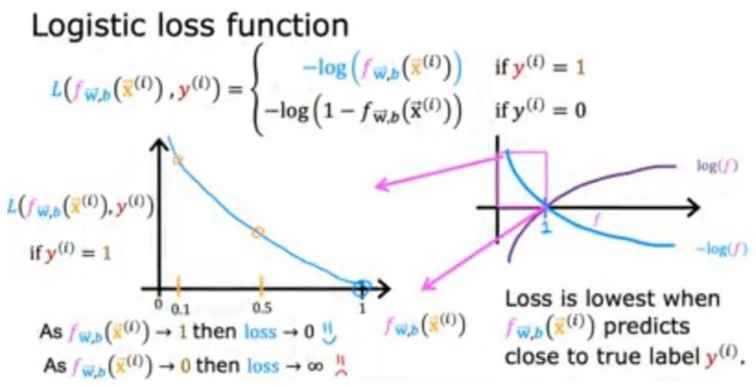
Logistic loss function

$$L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Now, f is the output of logistic regression. Thus, f is always between zero and one because the output of logistic regression is always between zero and one. The only part of the function that's relevant is therefore this part over here, corresponding to f between 0 and 1. Let's zoom in and take a closer look at this part of the graph. If the algorithm predicts a probability close to 1 and the true label is 1, then the loss is very small. It's pretty much 0 because you're very close to the right answer. Now continue with the example of the true label y being 1, say everything is a malignant tumor.

If the algorithm predicts 0.5, then the loss is at this point here, which is a bit higher but not that high. Whereas in contrast, if the algorithm were to have outputs at 0.1 if it thinks that there is only a 10 percent chance of the tumor being malignant but y really is 1. If really is malignant, then the loss is this much higher value over here. When y is equal to 1, the loss function incentivizes or nurtures, or helps push the algorithm to make more accurate predictions because the loss is lowest, when it predicts values close to 1.

Now on this slide, we'll be looking at what the loss is when y is equal to 1.



Logistic loss function

$$L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) & \text{if } y^{(i)} = 0 \end{cases}$$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 1$ then loss $\to 0$

As $f_{\vec{w},b}(\vec{x}^{(i)}) \to 0$ then loss $\to \infty$

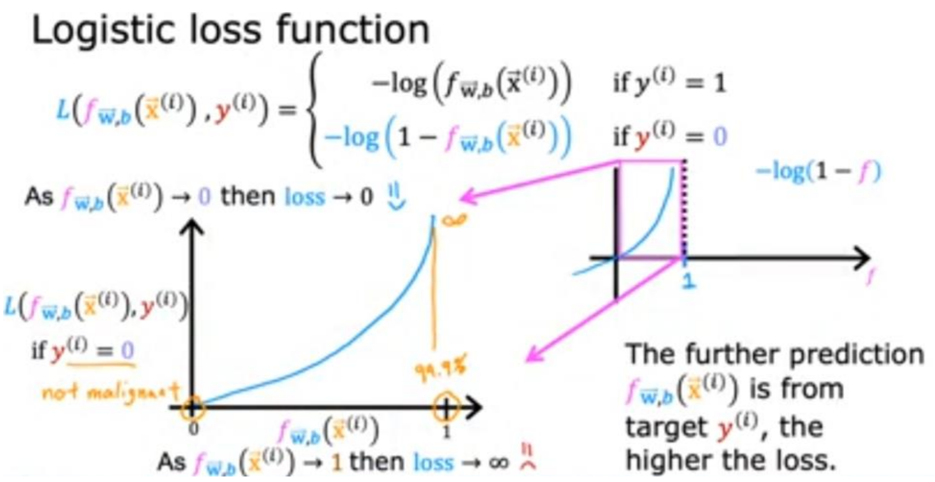
Loss is lowest when $f_{\vec{w},b}(\vec{x}^{(i)})$ predicts close to true label $y^{(i)}$.

Now on this slide, we'll be looking at what the loss is when y is equal to 1. On this slide, let's look at the second part of the loss function corresponding to when y is equal to 0. In this case, the loss is negative log of 1 minus f of x. When this function is plotted, it actually looks like this. The range of f is limited to 0 to 1 because logistic regression only outputs values between 0 and 1. If we zoom in, this is what it looks like.

In this plot, corresponding to y equals 0, the vertical axis shows the value of the loss for different values of f of x. When f is 0 or very close to 0, the loss is also going to be very small which means that if the true label is 0 and the model's prediction is very close to 0, well, you nearly got it right so the loss is appropriately very close to 0. The larger the value of f of x gets, the bigger the loss because the prediction is further from the true label 0.

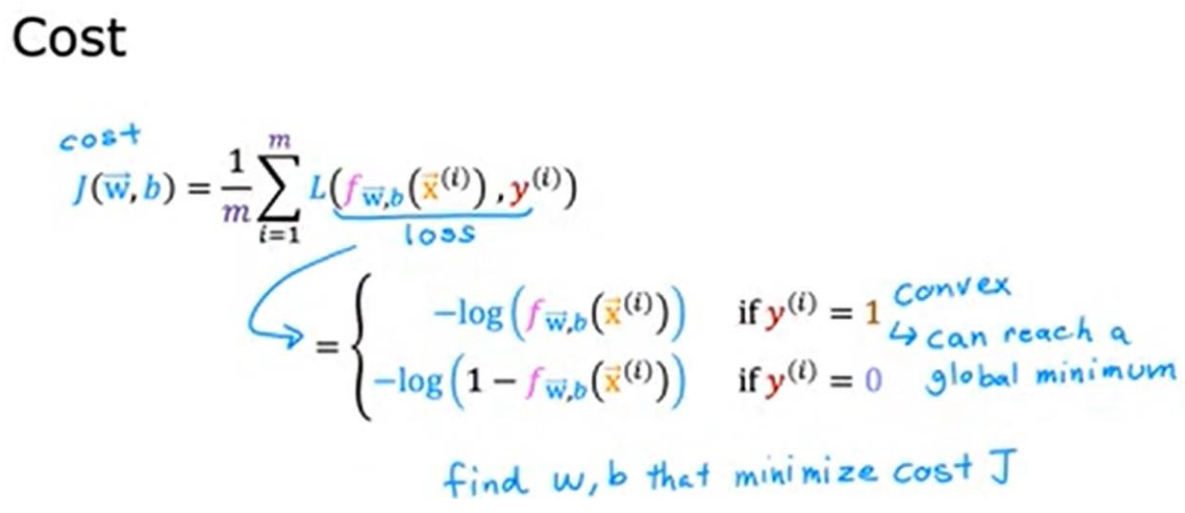
In fact, as that prediction approaches 1, the loss actually approaches infinity. Going back to the tumor prediction example just says if the model predicts that the patient's tumor is almost certain to be malignant, say, 99.9 percent chance of malignancy, that turns out to actually not be malignant, so y equals 0 then we penalize the model with a very high loss. In this case of y equals 0, so this is in the case of y equals 1 on the previous slide, the further the prediction f of x is away from the true value of y, the higher the loss.

In fact, if f of x approaches 0, the loss here actually goes really large and in fact approaches infinity. When the true label is 1, the algorithm is strongly incentivized not to predict something too close to 0.



In this video, you saw why the squared error cost function doesn't work well for logistic regression. We also defined the loss for a single training example and came up with a new definition for the loss function for logistic regression. It turns out that with this choice of loss function, the overall cost function will be convex and thus you can reliably use gradient descent to take you to the global minimum.

Proving that this function is convex, it's beyond the scope of this cost. You may remember that the cost function is a function of the entire training set and is, therefore, the average or 1 over m times the sum of the loss function on the individual training examples. The cost on a certain set of parameters, w and b, is equal to 1 over m times the sum of all the training examples of the loss on the training examples. If you can find the value of the parameters, w and b, that minimizes this, then you'd have a pretty good set of values for the parameters w and b for logistic regression.



Cost

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})}_{\text{loss}}$$

$$= \begin{cases} -\log(f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w},b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

convex
↳ can reach a global minimum

find w, b that minimize cost J