What we will be learning:

Supervised Learning (regression, classification)

Unsupervised Learning (clustering, afterwards)

Regression Model

^y stands for estimated value of y y = target value (unknown?)

f, function is also called THE MODEL

what is the math formula we're going to use to compute f?

f w,b (x) = wx + b OR DIRECTLY f(x)

Constructing a Cost Function

w,b: parameter/coeeficient/weight (it determines your function)

w = slope

(y^ - y) = error

We use **J** to refer to our cost function

We made a "squared error cost function". (Least squares estimation)

Cost functions differ from people to people, or depending on many things like our problem.

*goal of linear regression is to: minimize J (w,b) → w is an array, b is a constant

We saw how value of J differs depending on parameters, w and b. We need to try them to find values of w and b, that minimizes J. But we can't try them on our own, one by one.

SOLUTION: algorithms that will to this task for us

***GRADIENT DESCENT (and many many more)**

You can try to use it to minimize any function.

*for linear regression, with squared error cost function, you always get a bow shape

for cost (J) in the end

Start with initial values. (0 if convenient) Keep changing "parameters to try to

reduce J. THERE MAY BE MORE THAN 1 MINIMUMS

FOR FURTHER EXPLANATION, THINK OF THE EXAMPLE OF HIGHHILLS AND VISUALIZE IT

you may end up in different valleys, if there are multiple local minimums

*gradient descent continuously tries to get a lower value. So if you find a minimum,

you are done and have no place to go.

GRADIENT DESCENT

ALGORITHM --> $w = w - \alpha \cdot \frac{d}{dw} J(w,b)$ $b = b - \alpha \cdot \frac{d}{db} J(w,b)$

α = learning rate, positive number between 0 and 1 (it's small)

REPEAT THIS UNTIL CONVERGE (the point where w and b no longer changes)

***parameters are updated "*simultanously*"**

Use of derivative is explained in "G.D. Intuition" part. If your parabola is going right and up, then derivative is positive, so a number is being substracted from w. W decreases.
If parabola is going left and up, then derivative is negative; therefore a positive number is being added to w. W increases. This step happens repeatedly and you get closer to the optimum value of w.

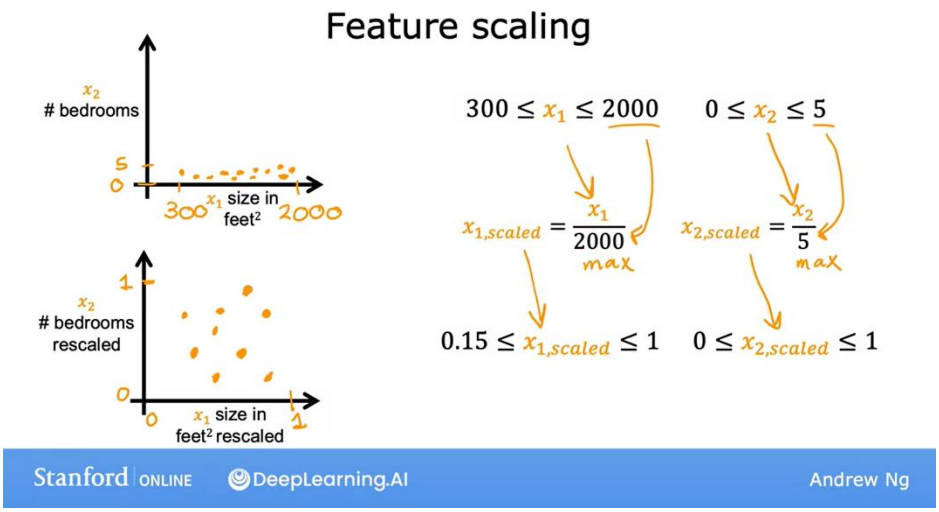
NEXT: WHAT IS "LEARNING RATE"? What happens when it's too small or too big?
How am i supposed to choose a better learning rate?

FEATURE SCALING

Parameters will most likely be in wide ranges and they will differ. One parameter will be in range(5) while other belongs to the range(2000).

So you need to scale them to work with both of them.

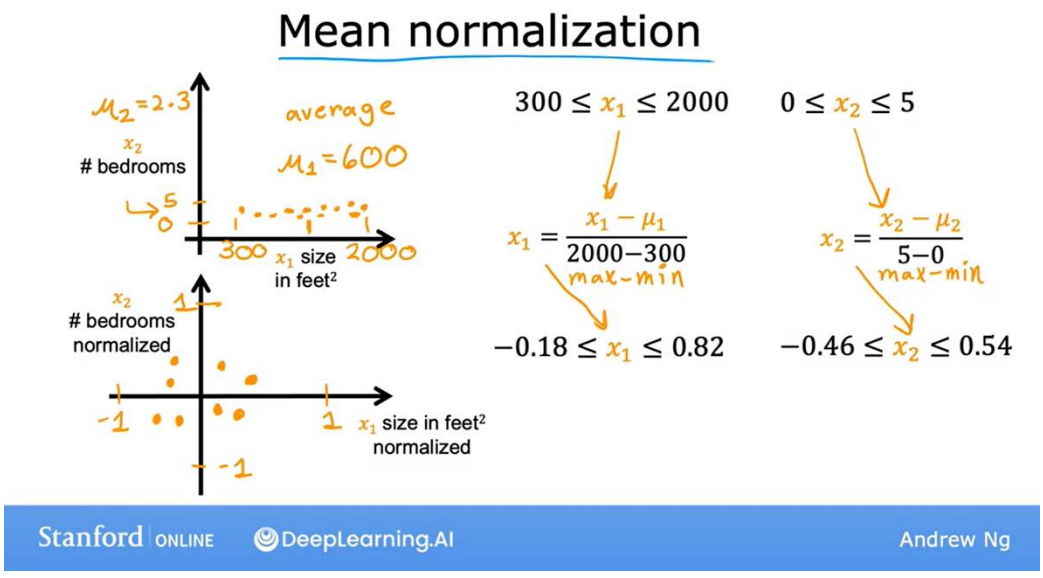
Solution: Divide the parameter with the max value. So maximum value of it is 1.



ANOTHER OPTION: **MEAN NORMALIZATION**

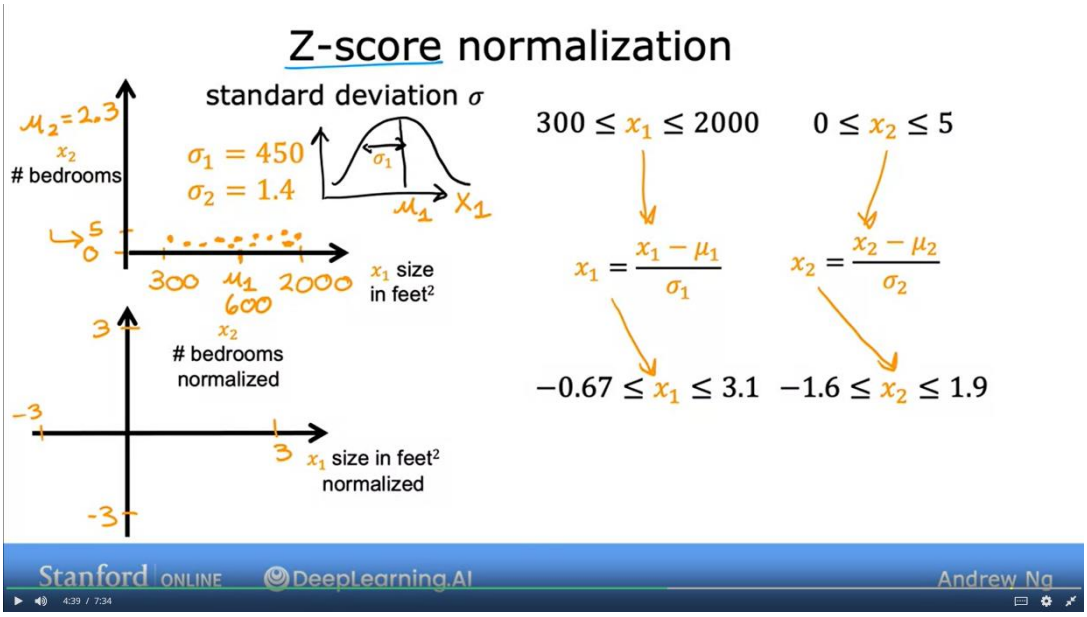
In this option, both parameters will be scaled to a state in which they are centered around (0,0). This option enables both negative and positive values.

1) Find mean value of the parameter.



ANOTHER OPTION: **Z-SCORE NORMALIZATION**

For that, you will need standard deviation. This method also makes data center around (0,0).



RULE OF THUMB in feature scaling: Aİm for -1 < x < 1 for each feature. These values can loose a little bit. It's alright.

Too small values are a problem, just like the ones that are too big.

Also 98,6 < x < 105 is a problem as well.

In the next one: How to recognize if the grad. desc. is descending?

Checking gradient descent for convergence

One of the key choices: Choice of learning rate

A two dimensional graph in which one axis is J and the other one is iterations: Learning curve

→ Desired behavior: J decreases. If it increases, even for a single iteration, sth is wrong.

After a number of iterations, J might cease to decrease, meaning it converges. Number of iterations needed to converge varies a lot. Telling this in advance is hard. So we use a graph.

ANOTHER WAY TO DECIDE IF THE TRAINING IS DONE:

AUTOMATIC CONVERGENCE TEST

Declare an "epsilon" value. If the amount J decreases is less than "epsilon", declare convergence.

An.NG: I usually use graphs, due to lack of insight in choosing the "epsilon" value.

Choosing the learning rate

Too small: Runs slow Too big: Will diverge

If J does not decrease continuously: Bug in your code or too large learning rate

Try different values. Increase them slowly, like

… 0,001 → 0,003 → 0,01 → 0,03 → 0,1 → 0,3 → 1 …

At some point you will find a value which is too large. Descend slowly from that value.

Optional lab:

Feature scaling, essentially dividing each positive feature by its maximum value, **or more generally, rescale each feature by both its minimum and maximum values using (x-min)/(max-min). Both ways normalizes features to the range of -1 and 1, where the former method works for positive features which is simple and serves well for the lecture's example, and the latter method works for any features.**

Mean normalization: $xi:=(xi-\mu i) \ / \ (max-min)$

Z-score normalization which we will explore below.

The scaled features get very accurate results **much, much faster! (Shown in lab with time values compared.)

**Any predictions using the parameters learned from a normalized training set must also be normalized.

**The result is that updates to parameters during gradient descent can make equal progress for each parameter.

Feature Engineering: Intuition about the problem is used to design new features, by transforming or combining original features.

e.g: we have width and depth. We can add an another feature like w.d (area)

*It makes learning algorithm's job easier and faster.

POLİNOMIAL REGRESSION

Linear re. İs pretty useful but mostly it won't suffice

***Quadratic equa. Eventually comes down. This behaviour is mosly wrong. Can use cubic maybe?

These are just a few examples.

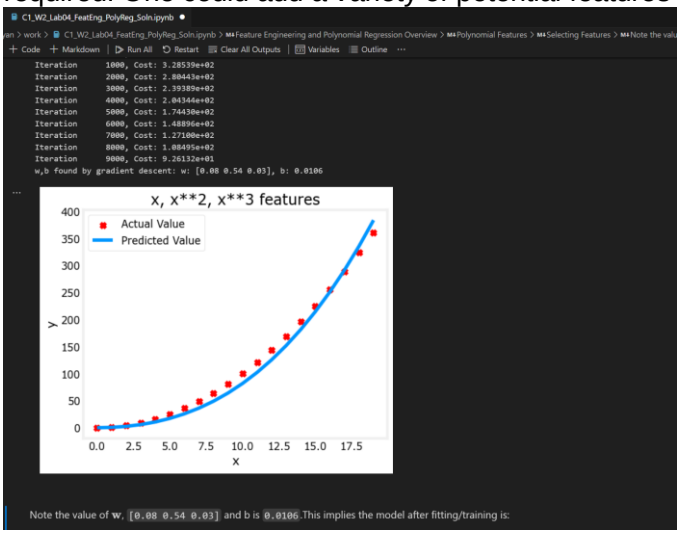
If you are using powers in the equation like this, feature scaling becomes increasingly important. That's because when you get a feature's square, it's range increases dramatically.

Getting square root of x might also work well in some occasions.

(in the 2. course we cover how to choose and scale appropriately)

Selecting Features

Above, we knew that an x^2 term was required. It may not always be obvious which features are required. One could add a variety of potential features to try and find the most useful



In the lab above, we used x, x^2, x^3 instead of just x^2. But gradient descent itself has emphasized that x^2 is the most suitable feature, by increasing it and decreasing others.

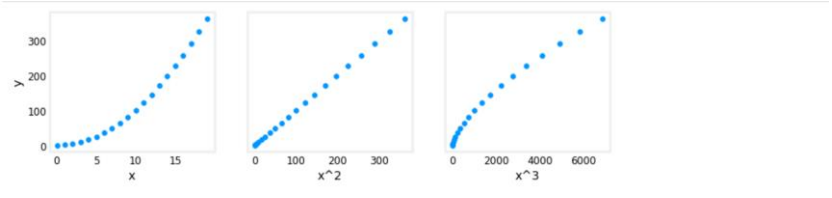
Gradient descent is picking the 'correct' features for us by emphasizing its associated parameter

Let's review this idea:

- Intially, the features were re-scaled so they are comparable to each other
- less weight value implies less important/correct feature, and in extreme, when the weight becomes zero or very close to zero, the associated feature is not useful in fitting the model to the data.
- above, after fitting, the weight associated with the x_2^2 feature is much larger than the weights for x or x_3^3 as it is the most useful in fitting the data.

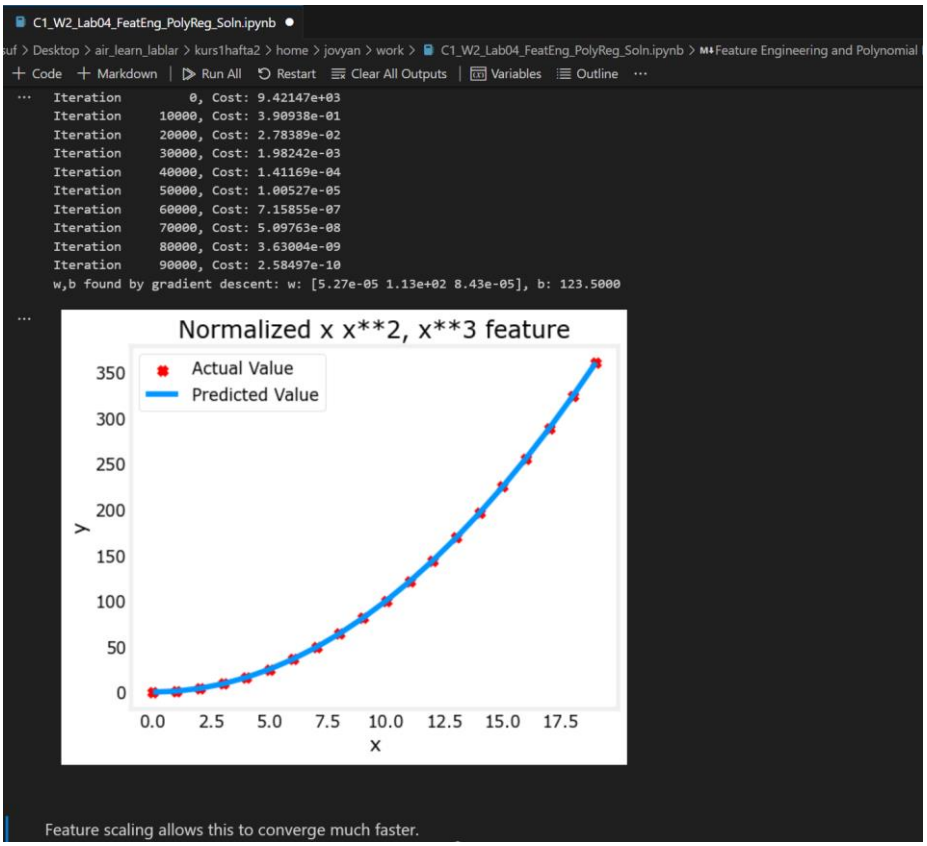
An Alternate View

Above, polynomial features were chosen based on how well they matched the target data. Another way to think about this is to note that we are still using linear regression once we have created new features. Given that, the best features will be linear relative to the target. This is best understood with an example.:
IN OTHER WORDS: More suitable features will show a more “linear like” graph. That’s because it’s weight increases and effects the cost function more.



Above, it is clear that the x^2 feature mapped against the target value y is linear. Linear regression can then easily generate a model using that feature.

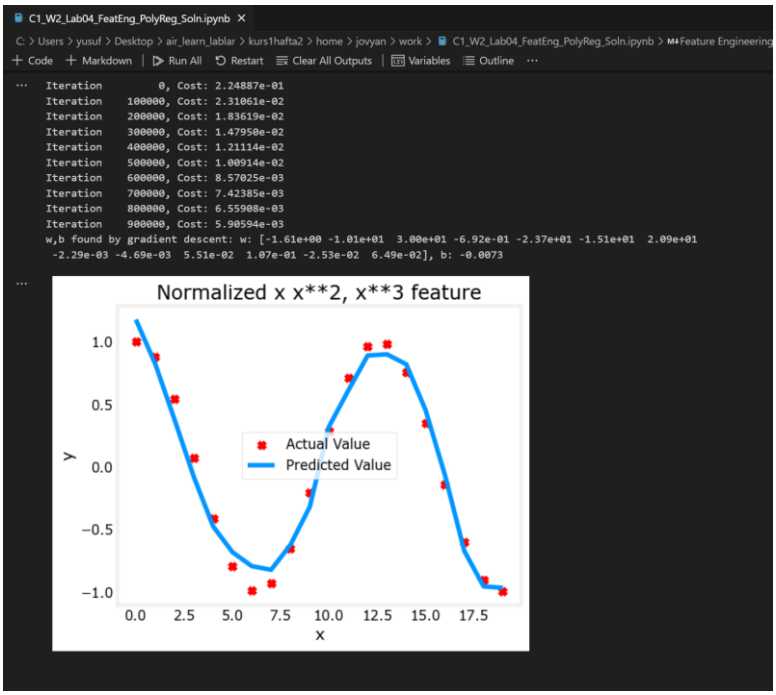
VERY IMPORTANT EXAMPLE ABOUT FEATURE SCALING



LOOK AT THAT RATE OF DESCEND

It converges soo much faster.

Note: With feature engineering, even very complex functions can be modeled.



Lessons From Programming Lectures

1) **Get familiar with your data.** Inspect your x,y and their shapes.
   Shape means number of training axamples in your dataset.
2) **Visualize your data**
   It is often useful to understand the data by visualizing it.
   For this dataset, you can use a scatter plot to visualize the data, since it has only two properties to plot (profit and population).
   *Many other problems that you will encounter in real life have more than two properties (for example, population, average household income, monthly profits, monthly sales).When you have more than two properties, you can still use a scatter plot to see the relationship between each pair of properties.
3) **Your goal is to build a linear regression model to fit this data.**
   With this model, you can then input a new city's population, and have the model estimate your restaurant's potential monthly profits for that city.
4) Compute cost, gradient descent, learning parameters using batch grad.desc.
   Recall batch refers to running all the examples in one iteration.

WEEK 3: CLASSSIFICATION

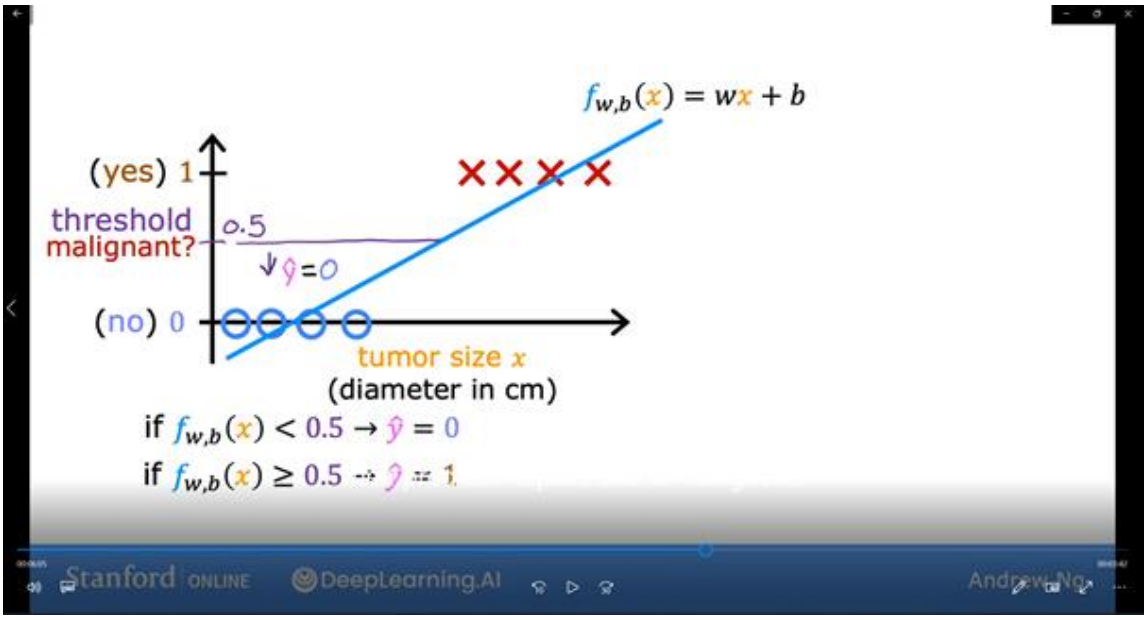
Y can only be certain values.

Yes or no → binary classification

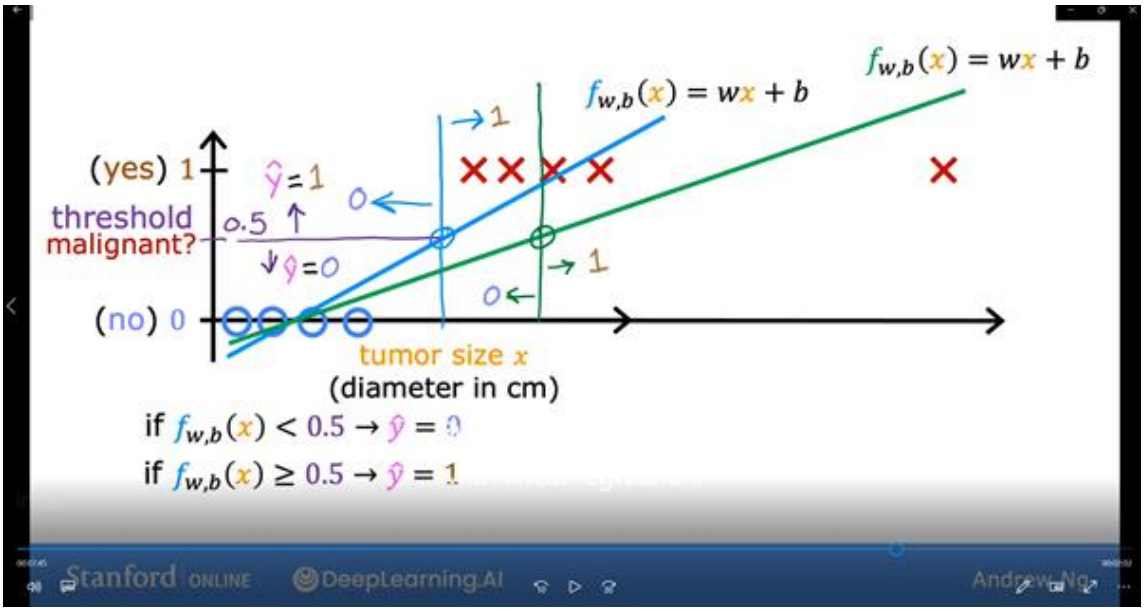
class and category mean basically same thing. Classes are “yesr or “no” or etc.

0 and 1 do not have semantic meanings. They refer to “absence” or “presence” of the thing we are looking for.

*Linear regression does not work in these problems.



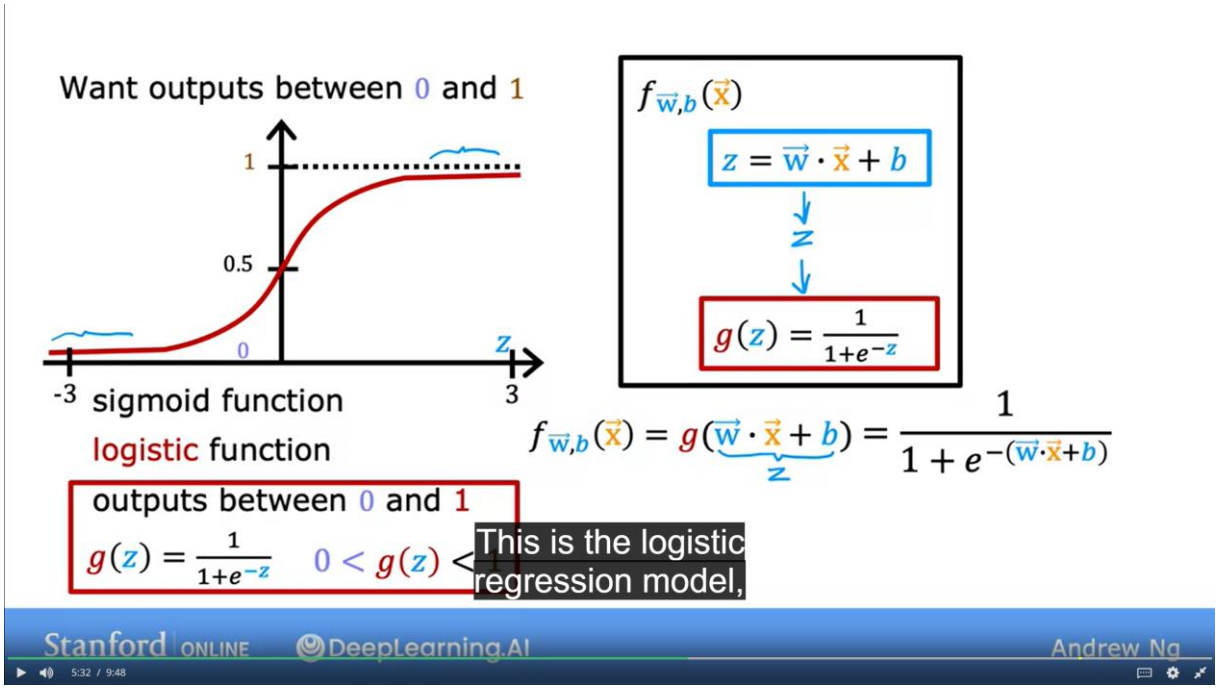
But still, in a situation you need to use it, you may apply rules, like picking thresholds and if number > 0.5, it's 1.



This might, however, work well in the first example and work poorly in the second one.

**Threshold = Decision boundary. In linear regression it changes in a poor way.

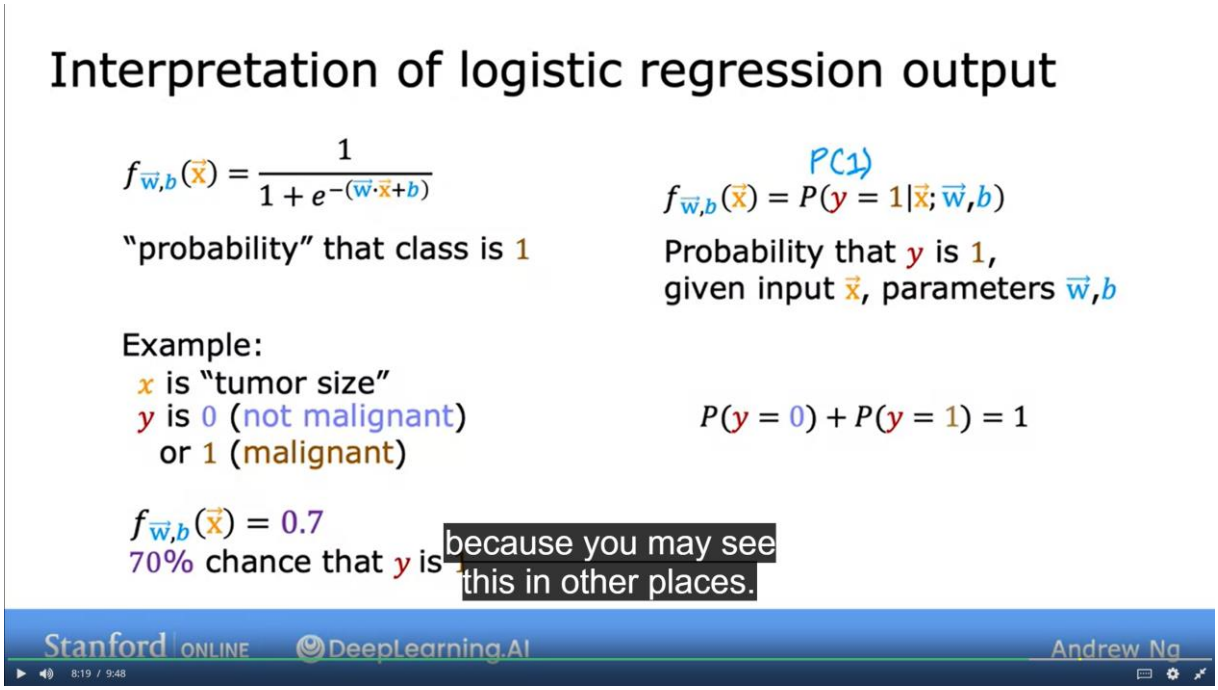
Sigmoid Function and Logistic Regression



Get z from linear function, pass it to the sigmoid function.

It gets x, outputs numbers between 0 - 1. Output actually means the probability for the class to be 1.

F(x) = 0.7 → %70 chance that y is 1. Y has got to be either 1 or 0.



Quote from Andrew NG: "For a long time, a lot of Internet advertising was actually driven by basically a slight variation of logistic regression. This was very lucrative for some large companies, and this is basically the algorithm that decided what ad was shown to you and many others on some large websites."

FROM LAB:

In the case of logistic regression, z (the input to the sigmoid function), is the output of a linear regression model.

- In the case of a single example, z is scalar.
- in the case of multiple examples, z may be a vector consisting of m values, one for each example.

Note: NumPy has a function called `exp()`, which offers a convenient way to calculate the exponential of all elements in the input array.

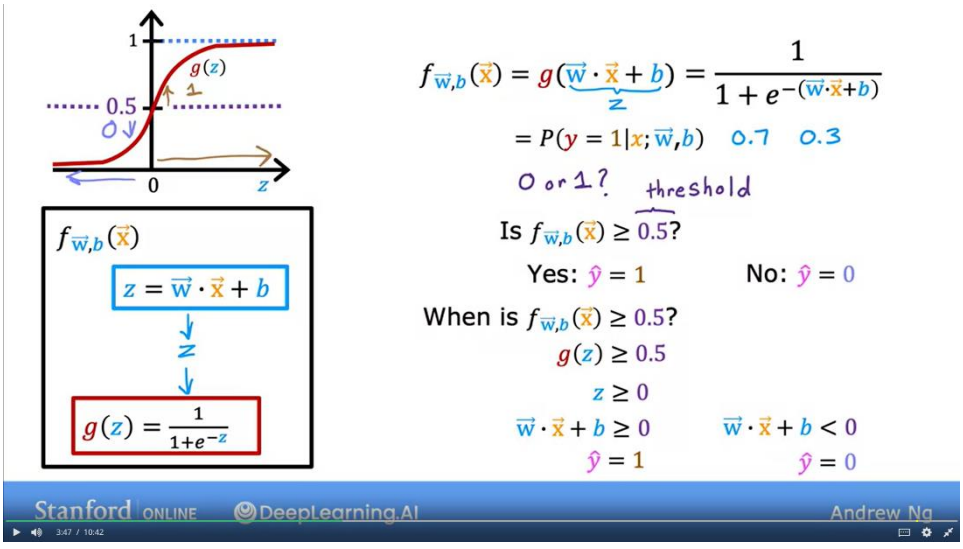
Decision boundary

1) compute z
2) apply simgoid function to the z

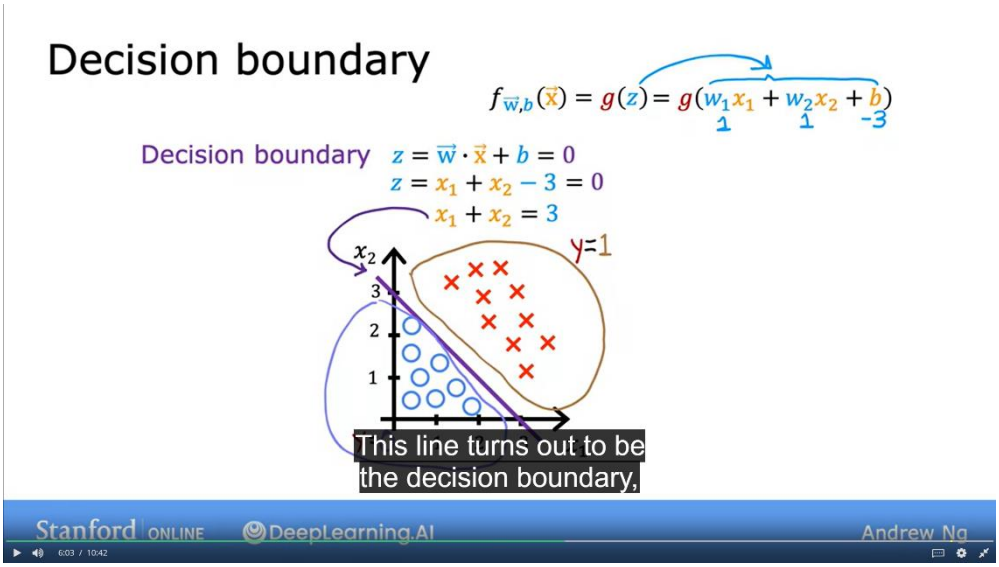
Threshold → 0.5 is a common choice. Whether the prediction will be 0 or 1, inspect the formulas, depend on the sign of z (w.x + b).

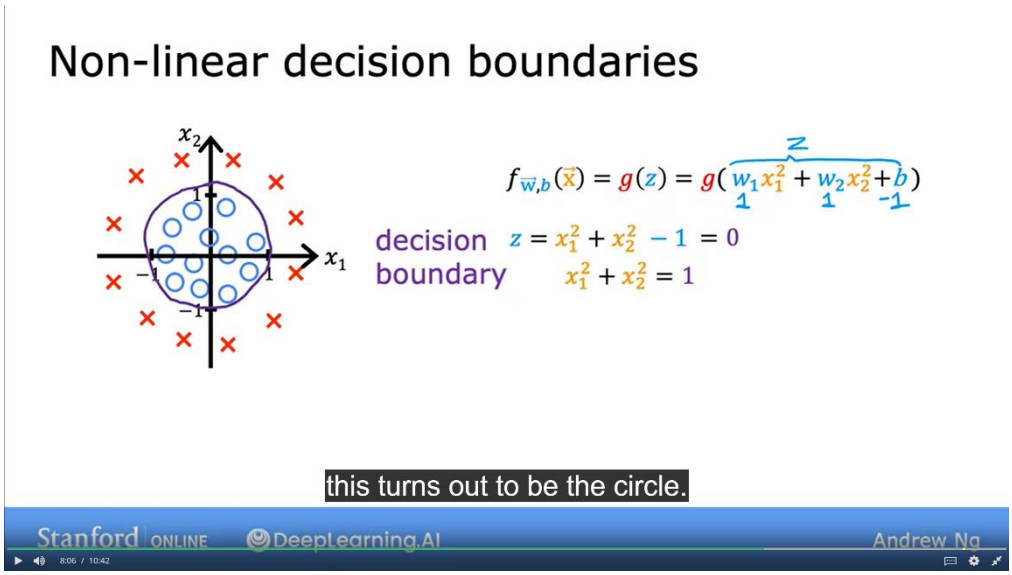
*Tanh can also be used for the interval (-1, 1). (CAN STILL REGULARIZE IT TO INTERVAL 0-1)

(it's what i did for my differential equations project)



On decision boundary, you are neutral about whether y is 0 or 1.





Just like polinomial regression, non-linear functions are used in logistic regression

You can also see more complex decision boundaries.

***If your parameters are like "x1, x2, x3…" your decision boundary will always be linear.**

*Let's say you are creating a tumor detection algorithm. Your algorithm will be used to flag potential tumors for future inspection by a specialist. What value should you use for a threshold?

NOPE
High, say a threshold of 0.9?

YUP
Low, say a threshold of 0.2?

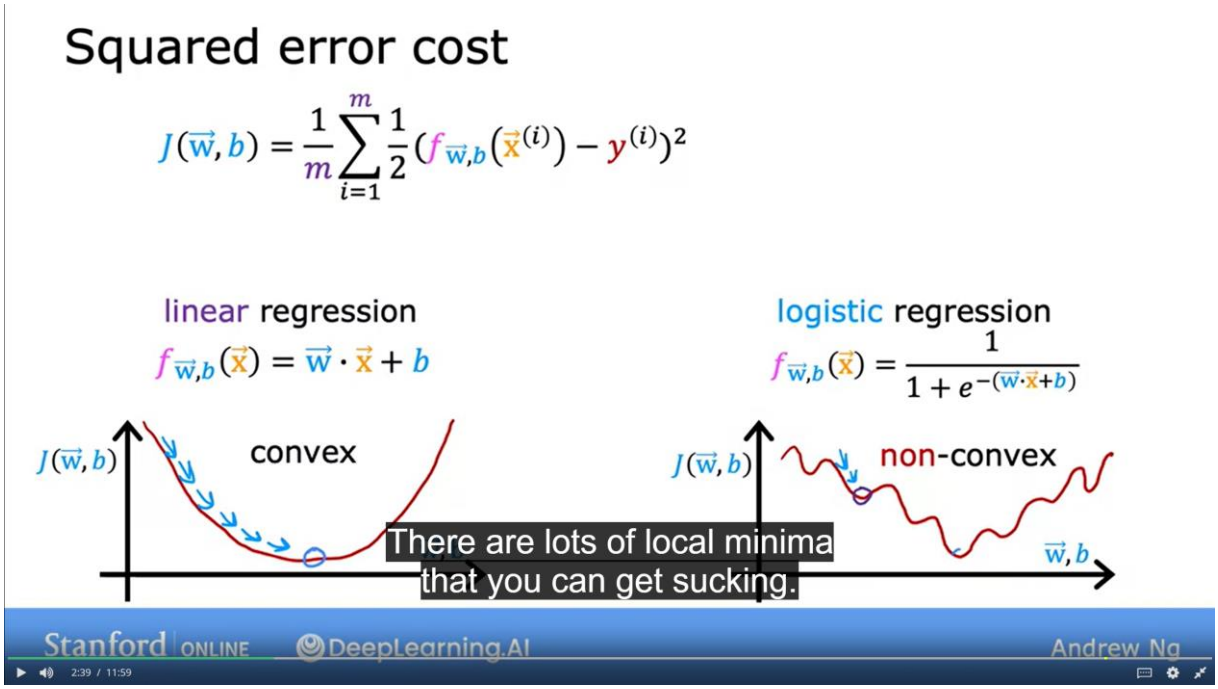
Correct: **You would not want to miss a potential tumor, so you will want a low threshold.** A specialist will review the output of the algorithm which reduces the possibility of

a ‘false positive’. The key point of this question is to note that the threshold value does not need to be 0.5.

As we've seen in the lectures, by using higher order polynomial terms (eg: $f(x)=g(x_{20}+x_1-1)$, we can come up with more complex non-linear boundaries.

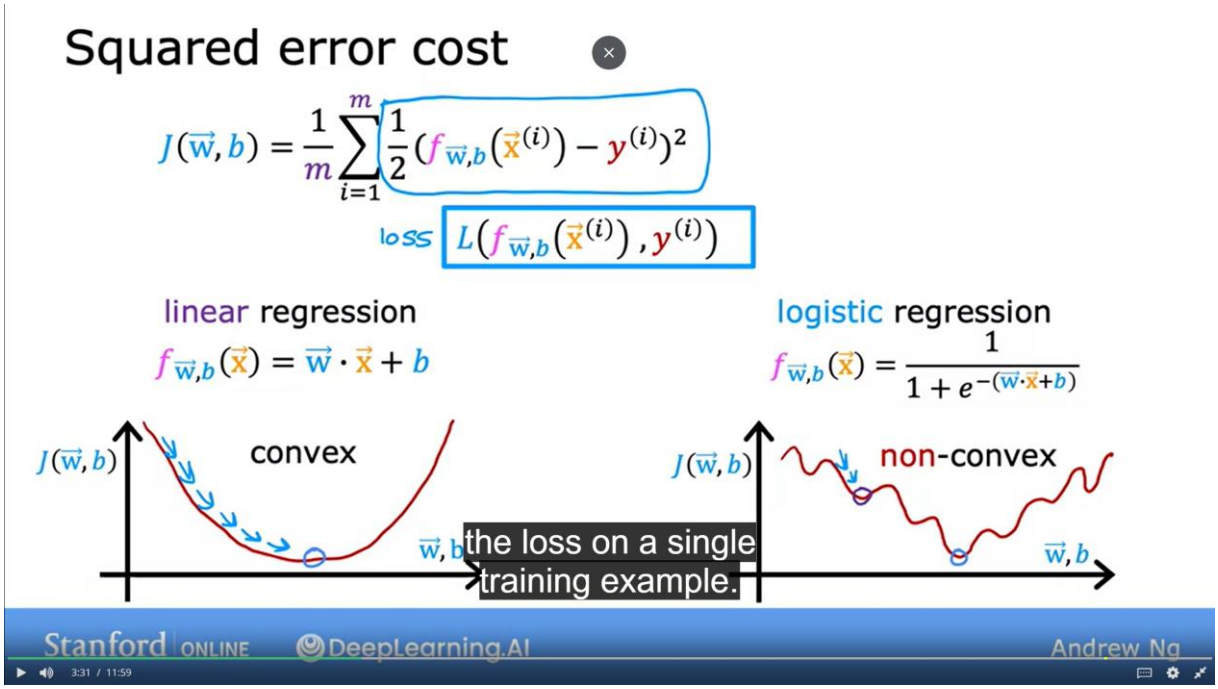
Cost function for logistic regression

Mean squared based cost function does not work for log.reg.

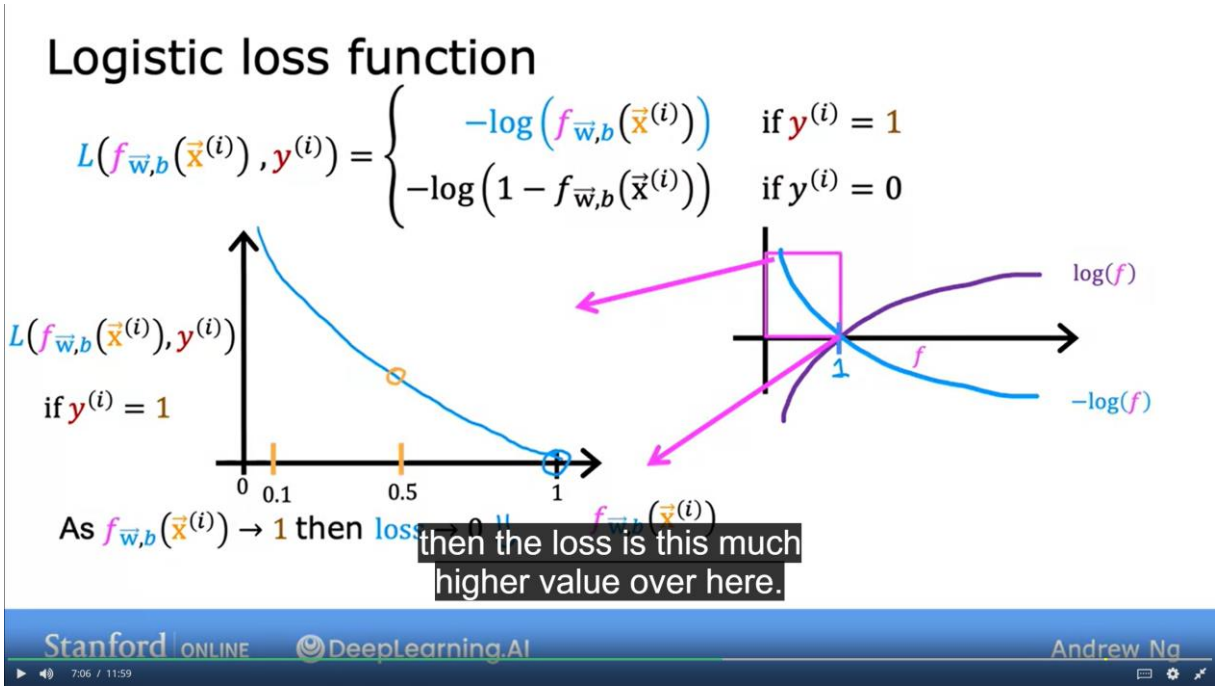


Loss: Is the cost function working on a single training example. → L

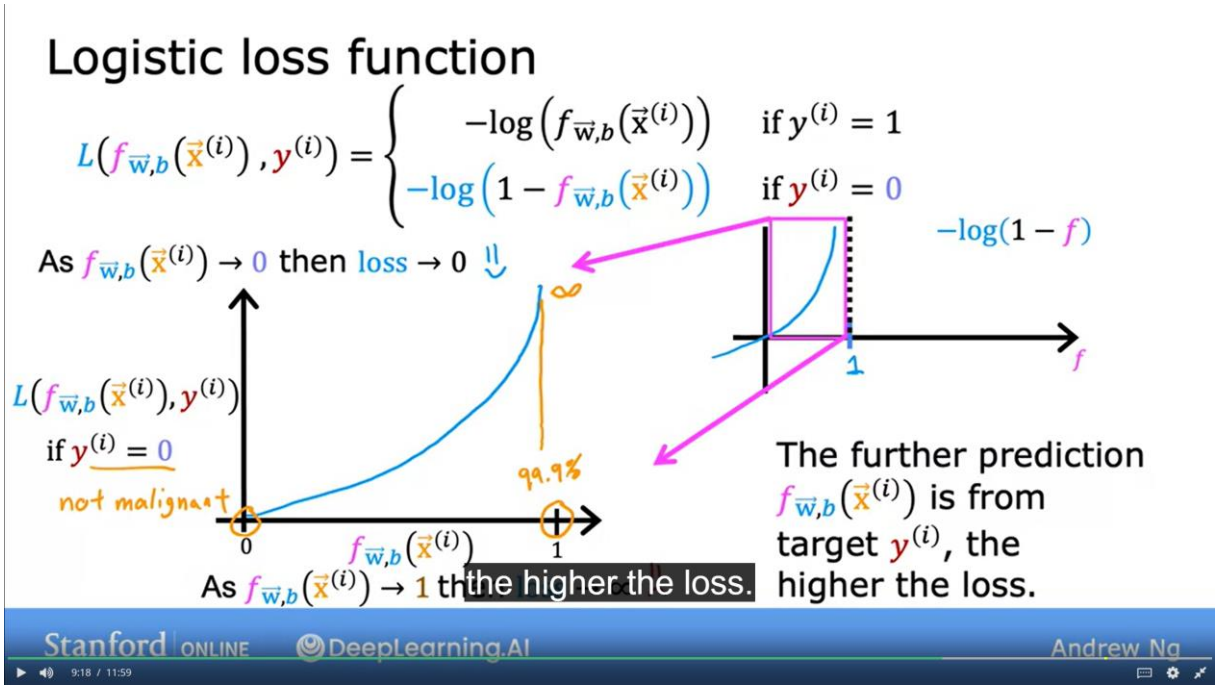
Cost function is the mean value of loss functions. **Loss function measures how well you are performing on one training example. Their sum gives you the cost value(entire training set).**

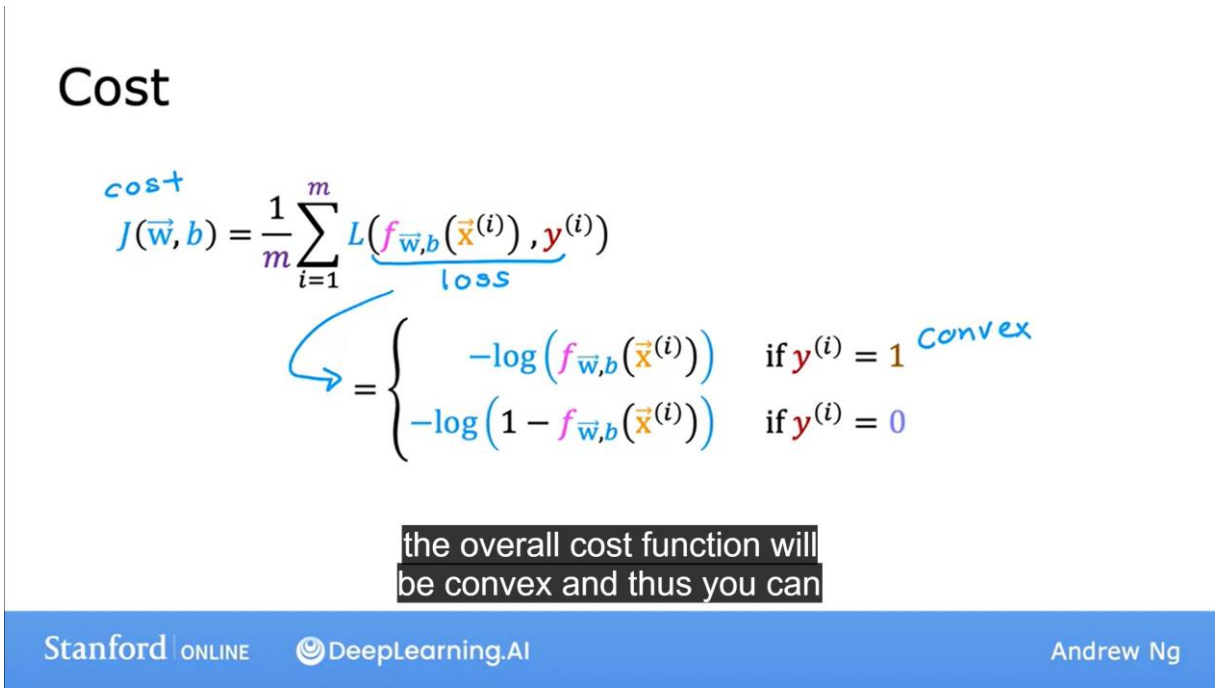


*In loss function, we use log.

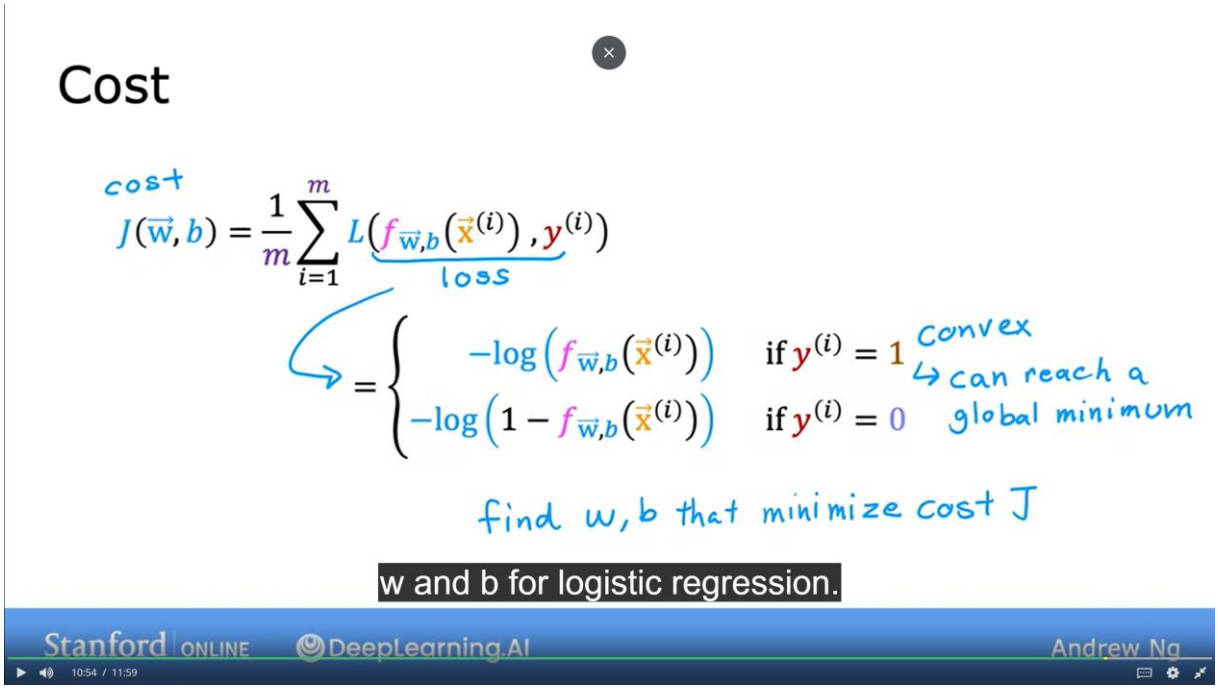


Logaritmic function provides the following: If predicted label is wrong, loss of that example is a lot higher than the desired amount.





This choice of loss function makes that cost function is convex. Therefore gradient descent can find the global minimum. (**Proving that it's convex is beyond the scope of this course**)



“”””

Loss is a measure of the difference of a single example to its target value while the **Cost** is a measure of the losses over the training set

“”””

The loss function above can be rewritten to be easier to implement.

$$loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), y^{(i)}) = (-y^{(i)} \log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right)$$

This is a rather formidable-looking equation. It is less daunting when you consider $y^{(i)}$ can have only two values, 0 and 1. One can then consider th
in two pieces:
when $y^{(i)} = 0$, the left-hand term is eliminated:

$$\begin{aligned} loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), 0) &= (-(0) \log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - (1 - 0) \log\left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) \\ &= -\log\left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) \end{aligned}$$

and when $y^{(i)} = 1$, the right-hand term is eliminated:

$$\begin{aligned} loss(f_{\mathbf{w},b}(\mathbf{x}^{(i)}), 1) &= (-(1) \log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) - (1 - 1) \log\left(1 - f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) \\ &= -\log\left(f_{\mathbf{w},b}\left(\mathbf{x}^{(i)}\right)\right) \end{aligned}$$

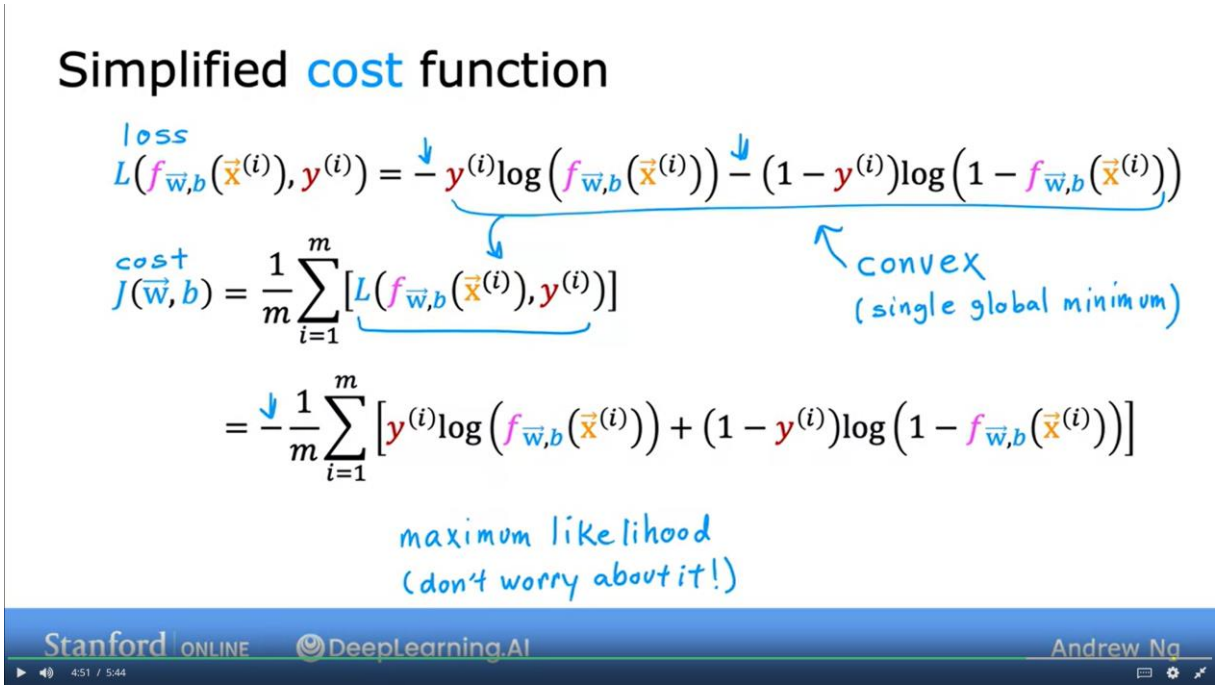
Simplified Cost Function for Logistic Regression

A better way to write loss and cost so the implementation is easier with gra.des.
It’s the formula on top. Y can only be 0 or 1, so these values can make the other part of the formula disappear.

Multiply the “y=1” part with y and the “y=0” part with (1-y). Each value makes the other part be 0.

Now we write the simplified Cost function. (below)

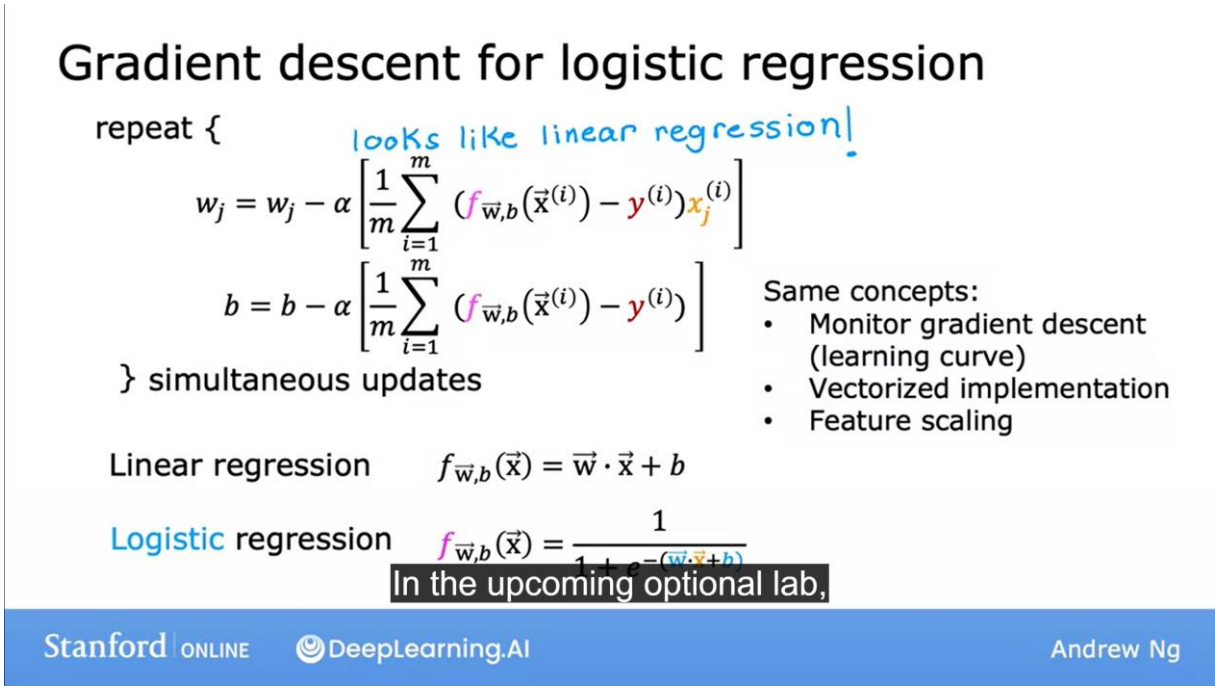
This formula is derived from statistics, using:
maximum likelyhood estimation. (used to find optimum parameters)



Now that we know these, let's learn how to implement gradient descent with this algorithm.

Our goal: Find parameters w, b. Way to do so: Use gradient descent

Form of the gradient descent algorithm is the same as linear regression. WHAT CHANGES IS THE DEFINITION OF THE FUNCTION F(X):



Time for the labs.

The problem of overfitting

Next videos: There is a regularization technic to minimize this problem.

Overfitting: Model fits for the training data **more than needed**. It learns the noise etc. And this causes the model to perform poorly in test data.

You can even find parameters that make Cost equal to 0. Because there is no error. IT'S BAD THOUGH.

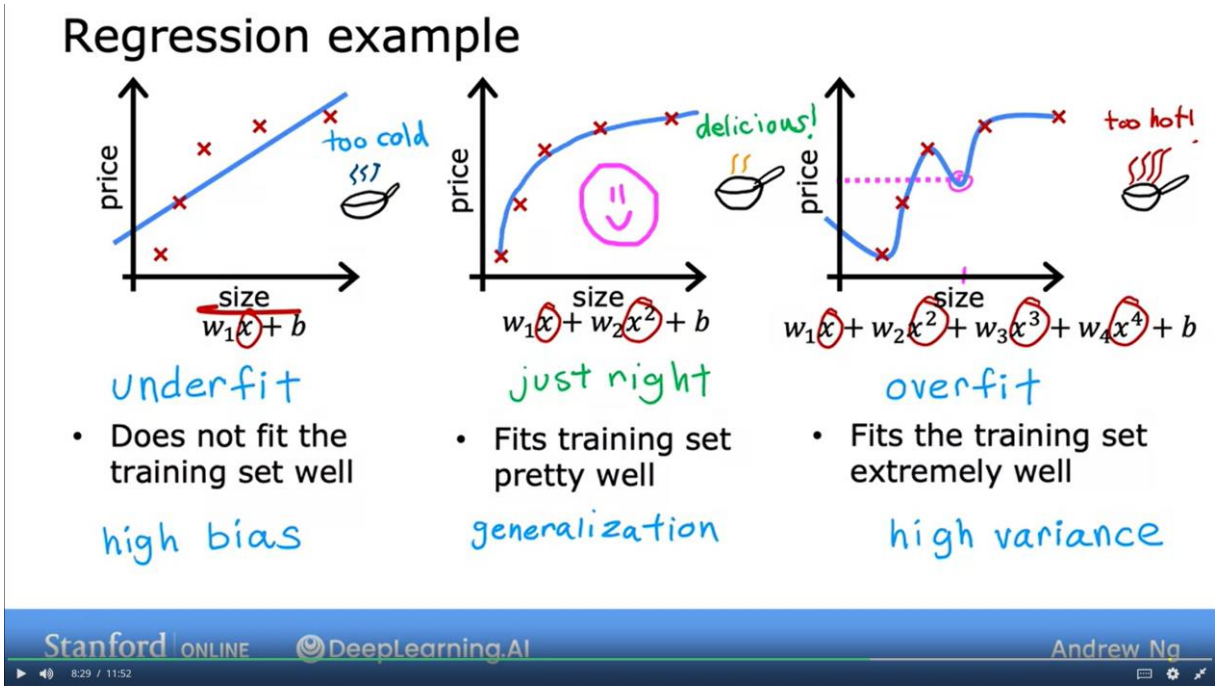
Model has high variance: If even one point of data was a bit different, function that has been fit might have been completely different.

Underfitting: Model is unable to fit. Might be too simple for example. Model has high bias.

If you succeed: Model is generalized.

BİAS: Model is unable to capture features of the data.

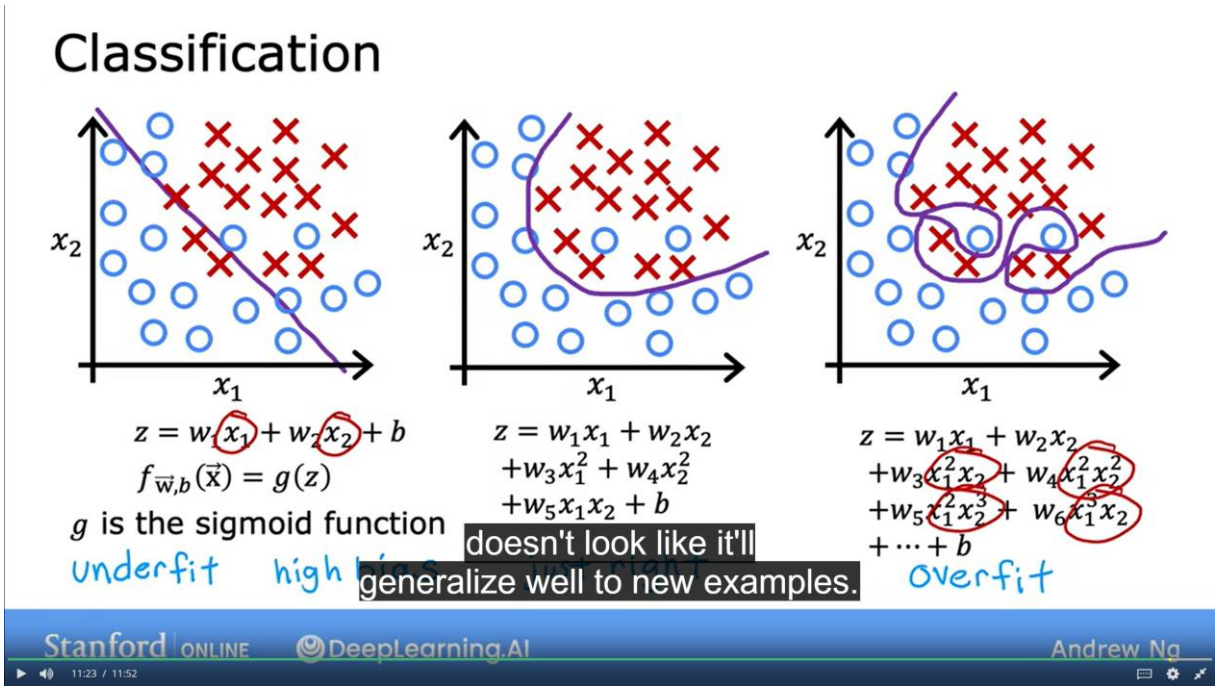
e.g: you assumed data is linear, you try to fit a line but data is more complicated.



HAVING TOO MUCH FEATURES CAN CAUSE OVERFITTING.

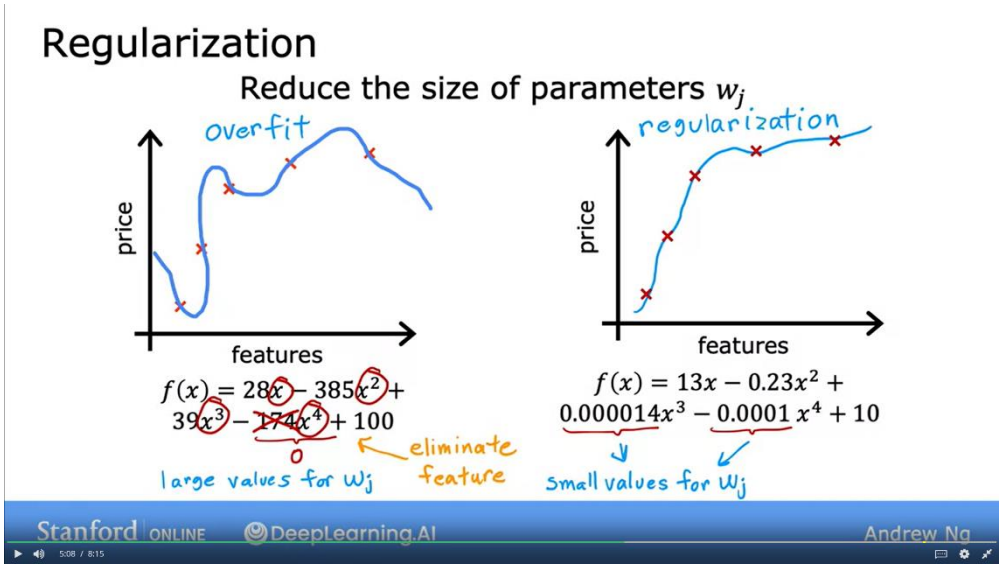
Too few can cause underfitting.

*Overfitting can occur in classification as well.



ADDRESSING OVERFITTTING

1) Collect mode training data: This makes the function less wiggly. This way you can still use more features and still do fine.
   You can not always get more data.
2) See if you can use **fewer features**: Having a lot of features but insufficient data might result in overfitting because model can not generalize with that less data. Try using less features (subset of features). This is called **feature selection. You use your intuition for this.** Though by using this, you are throwing away some information and possibly harming your performance. **COURSE 2 INVOLVES FEATURE SELECTION ALGORITHMS.**
3) **REGULARIZATON**



- Regularization lets you keep your features, but it prevents your features from having too much effect, which is what sometimes causes overfitting.
- We generally regularize parameters w1, w2, … wn. Regularizing b does not chance much. Andrew: “I usually dont”

Summarizing:

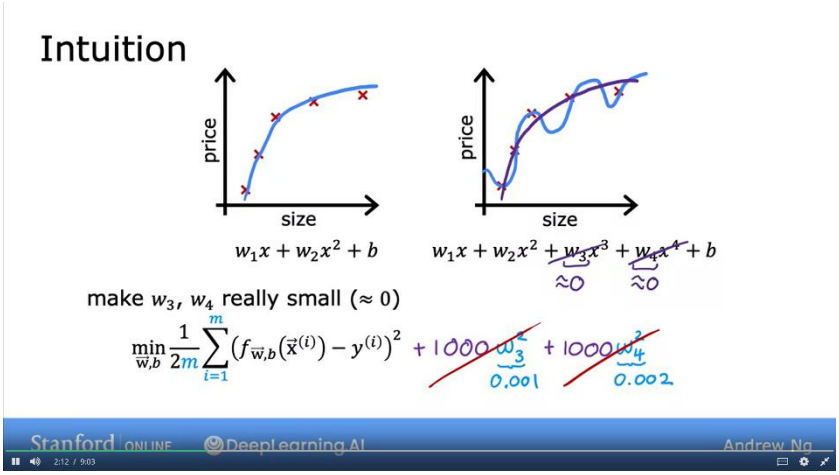
1) **Collect more data**
2) **Select features**
3) **Reduce size of parameters → regularization**

3. option is sth we almost always use for learning algorithms, especially for neural networks.

Note: Regularization basically makes everything better against overfitting. To sum it up, we implemented it with linear regression and logistic regression.

Cost function with regularization

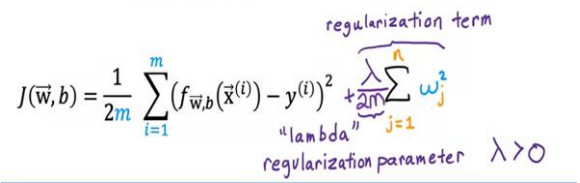
Situation: You fit a function with many features.



Logic: Add it to the cost function for it to be minimized.

****You don’t know which features to penalize** → Do it for all of them.

LAMBDA → regularization parameter. **We don’t penalize “b”. → lambda / 2m * b^2



**Lambda → Chooses how important reg. term is for the cost function.

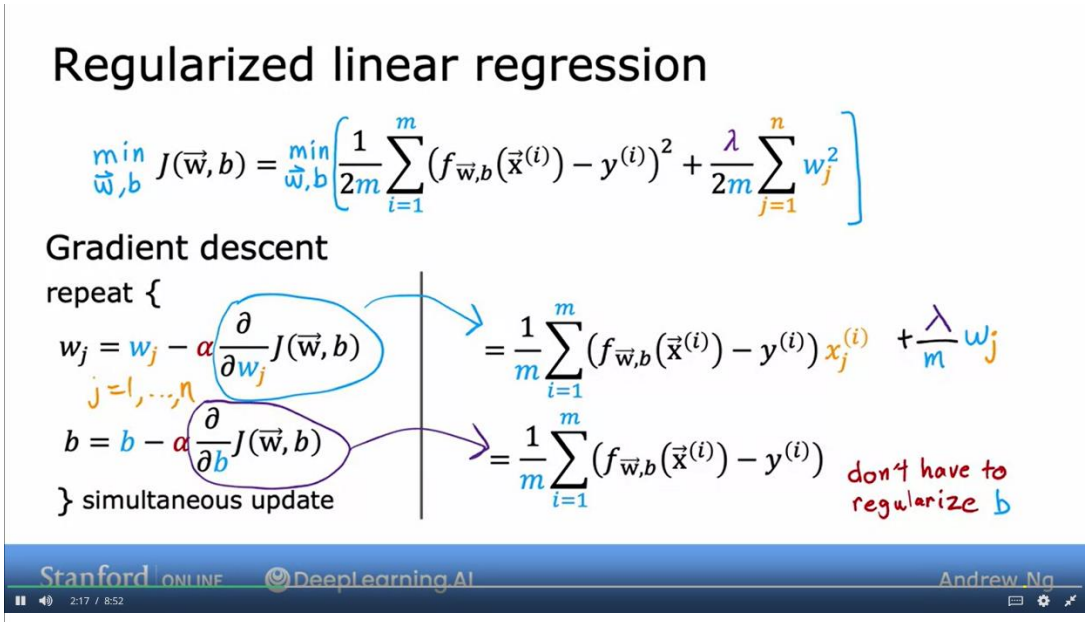
If lambda if 0, you don't regularize, it's not prone to overfitting. When it's too much, w's become nearly 0 and f(w,b) becomes b, fitting a straight line and underfitting.

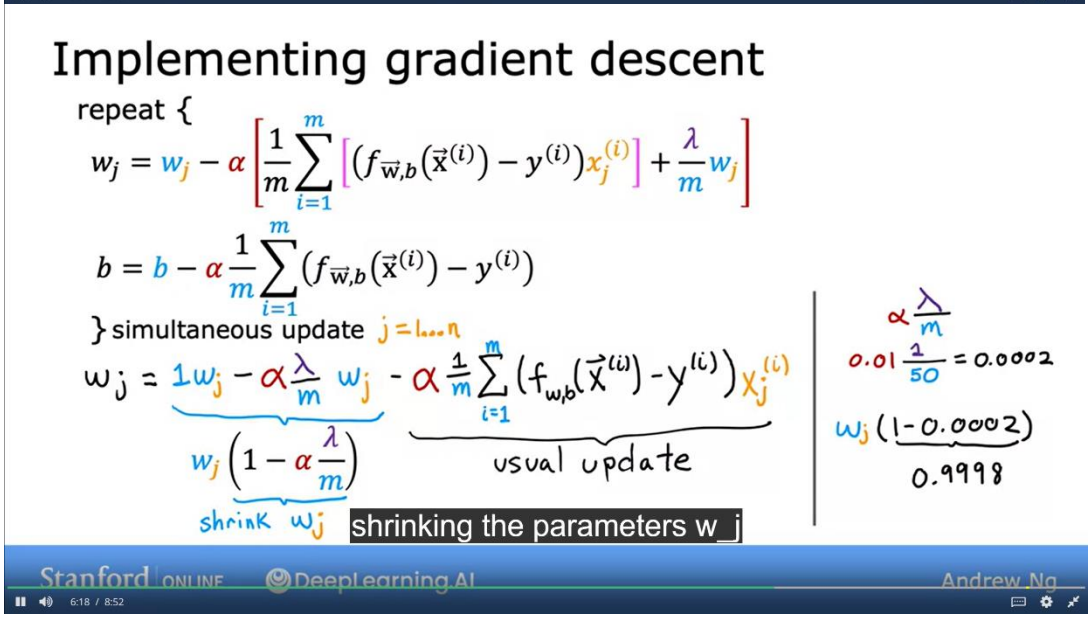
Choose a value in between.

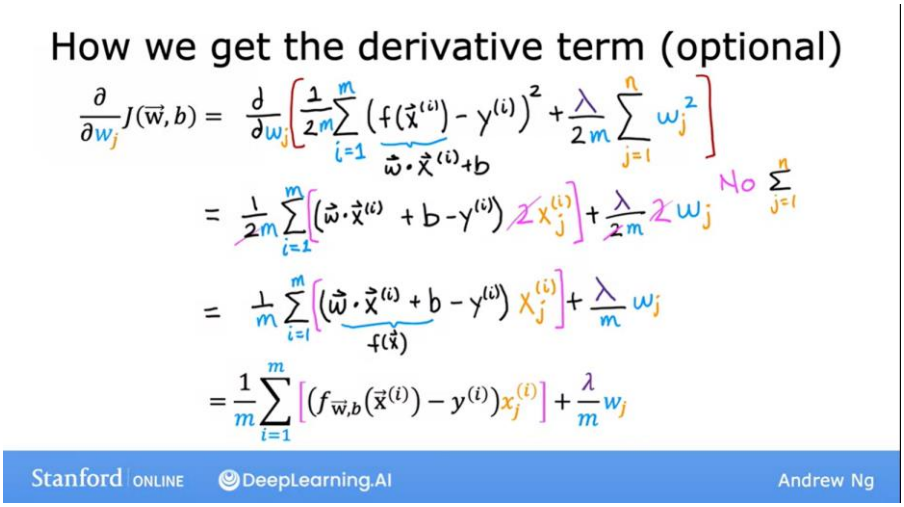
REGULARIZATON IF LIN. AND LOG. REGRESSION

The formula we used is only for the first term. (repeat until converge)

WHEN WE ADD THE REGULARIZATION TERM, PARTIAL DERIVATIVE OF THE FUNCTION CHANGES (and only that). We add a term for lambda.







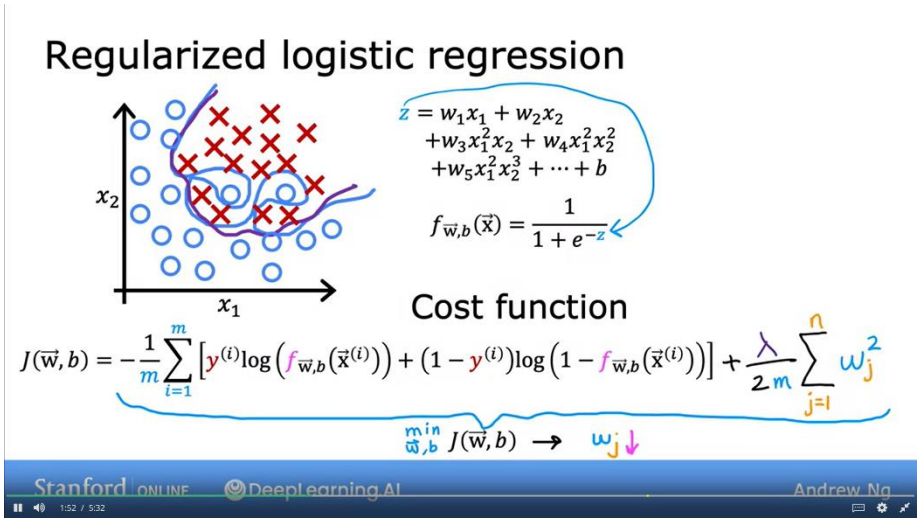
*****IN DERIVATIVE FORMULA, LAMBDA TERN DOES NOT HAVE "SUM" OPERATOR.**

We are already doing it for all the features.

Now for Log. Reg:

Again, only computing the gradient will change. Main goal is to, again, prevent overfitting.

**** We add the regularization term to cost function of logistic regression.**



To implement gradient descent, formula stays the same. We just add the reg. term. So partial derivatives of w_i change a bit. (EQUATION IS THE SAME AS LIN. REG. Only cost function changes.)

*Preventing overfitting is a valuable job.

***Last lab is pretty important.**

*Second course includes deep learning.

NOTE: Increasing λ will result with decrease in parameters w_i.

PRACTICE LAB 2 → LOGISTIC REGRESSION

1) Explore the data: values, dimensions, visualization etc.
2) **In sigmoid function, parameter z can always be an array of numbers. So to compute e^-z, use np.exp instead of math.exp.**
3) If X equals feature values of all data points, we use
   z = np.dot(X[i], w) + b to get dot product of i.th sample and parameters w_i.
4) Computation of gradients are the same as linear regression.
   a. Compute value of sigmoid function (takes .dot as input)
   b. Loop over data points and compute gradients
5) Fit the model. → Parameters give you the decision boundary.

6) In second part, in another problem, we added feature engineering. We added every possible feature up the 6.th power of x_1 and x_2.

   FEATURE MAPPING

   THIS INCREASED THE DIMENSIONS FROM 108*2 TO 108*27. **THUS, MODEL BECAME PRONE TO OVERFITTING.**
7) Just add regularized_cost term for the cost function. That's all.