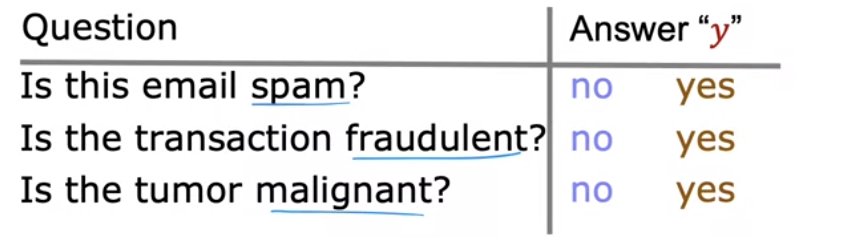
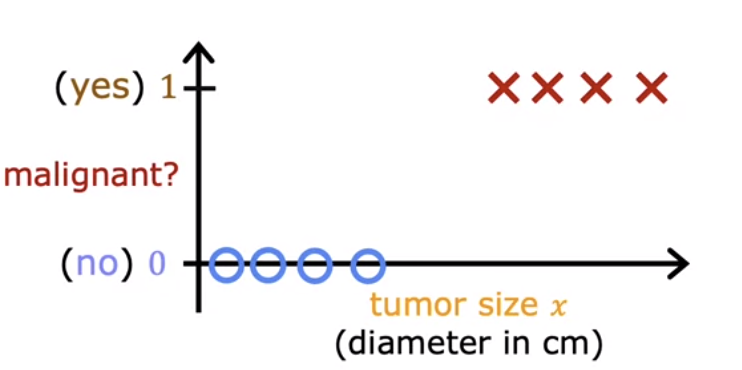
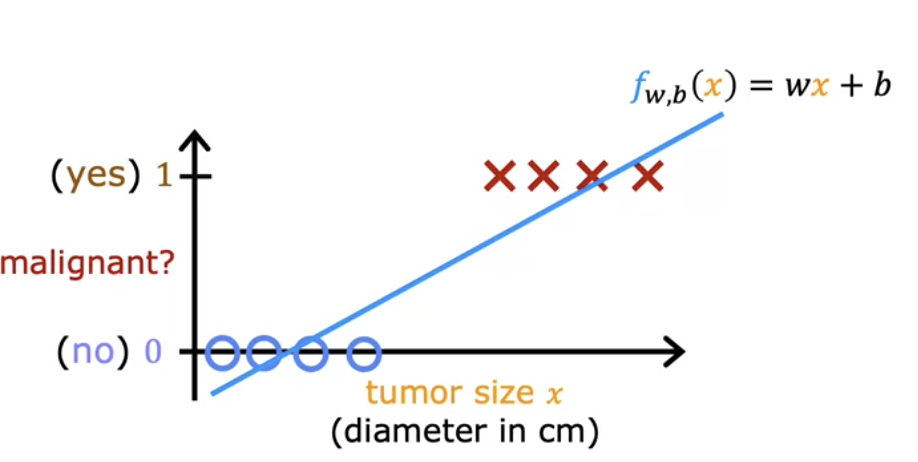
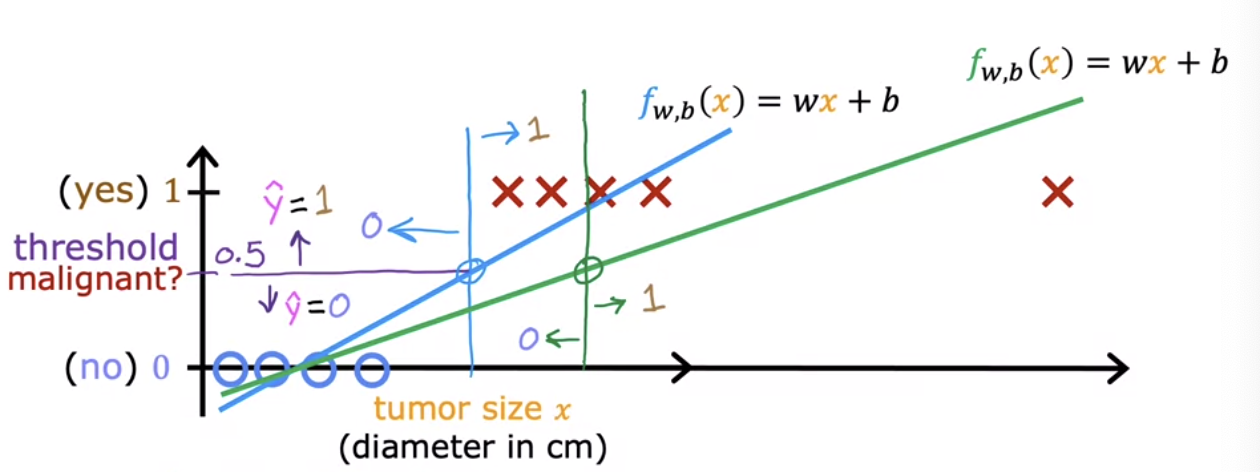
# Week 3: Classification

## Classification with Logistic Regression

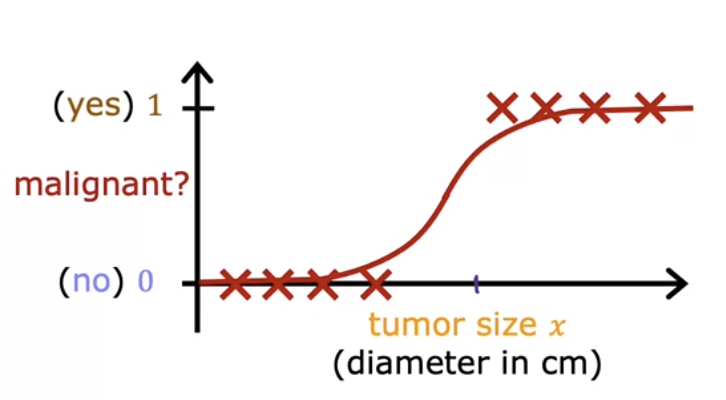
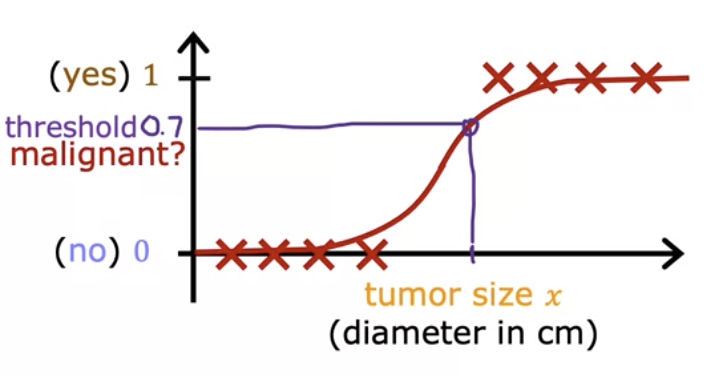
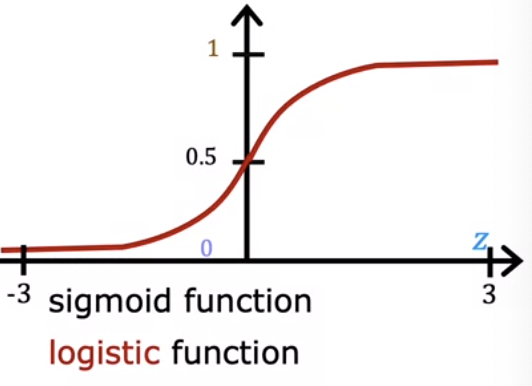
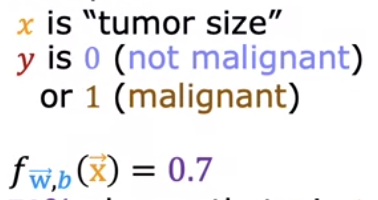
### Motivations

* + In classification, y can only be a small range of possible values
    - In this case, *y* can only be one of two values → **binary classification**
    - Common y values
      * Could be no or yes/false or true/0 or 1 (all of these refer to the same things
        + 0 is referred to as the negative class, while 1 is referred to as the positive class
  + In classification, class = category (terminology)
    - Can be used interchangeably
  + Could you apply linear regression to classification algorithms?
    - Example:
      * Could have a best fit line like this:
        + Linear regression does not only predict the values of 0 and 1; it predicts all values → our goal though is to predict categories

Could use a threshold approach

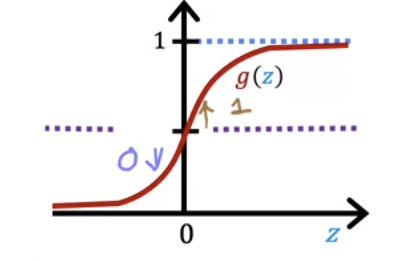
* + - * But this doesn’t work for all training sets
        + If we add one more training example to the right of the graph, then doesn’t work anymore as some malignant training examples will be classified as 0 as the output (some values are misclassified)

### Logistic Regression

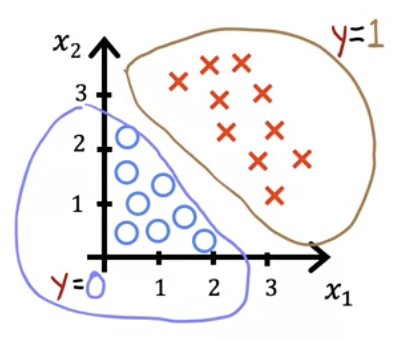
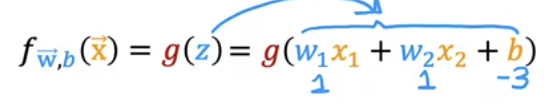
* + Classification algorithm
  + Example Data Set:
    - Fits an S sized curve to the data → different threshold than linear regression
  + Sigmoid Function / Logistic Function
    - Outputs values between 0 and 1
      * + When *z* is large → the value of *g(z)* will be close to 1
        + When z is small → the value of *g(z)* will be close to 0
  + Logistic Regression Algorithm
    - Start with
      * → pass it to
        + → when combined they give you the logistic regression model
    - Interpeting the output
      * “Probability” the class is 1
        + Equation means the probability *y = 1* given *x* and the parameters
      * Example: 
        + Model thinks there is a 70% chance that *y* is 1 for this patient

30% chance that *y* is 0 for this patient

### Decision Boundary

* + You want the learning algorithm to decide whether to assign the y value 0 or 1
    - Could set threshold
      * Common choice to set threshold = 0.5

→

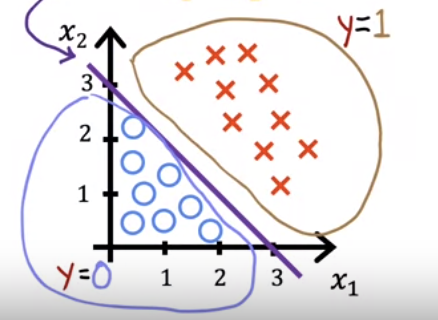
* + How does the model make a prediction?
    - Example with two features:
      * → Will use this model:
        + Pretend: 

Look at when

This line when z = 0 is referred to as the **decision boundary**

Neutral about y being 0 or 1

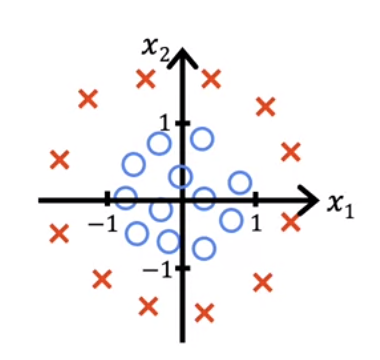
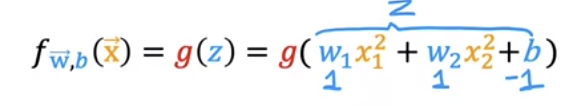
For what x values is ?



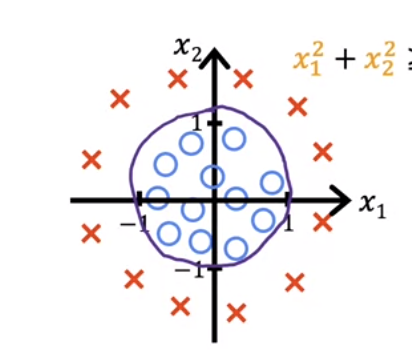
→ Results in the purple line (decision boundary)

If to the right of the line → algorithm predicts 1

If to the left of the line → algorithm predicts 0

* + Non-linear decision boundaries
    - You could use polynomials to get decision boundaries
      * Set
        + Pretend 

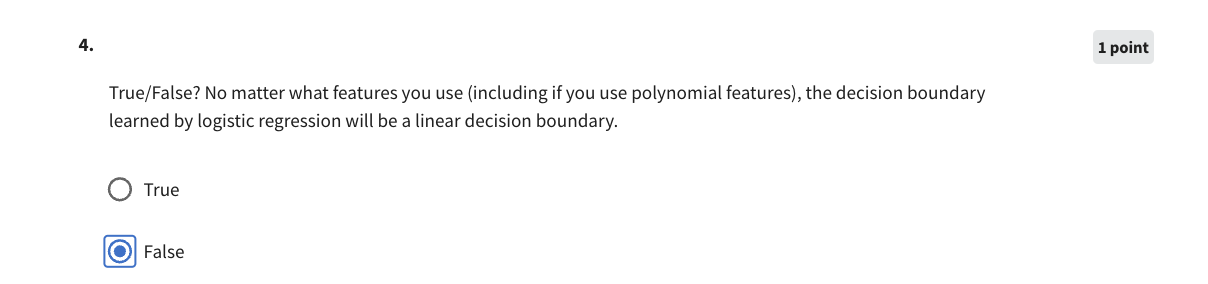
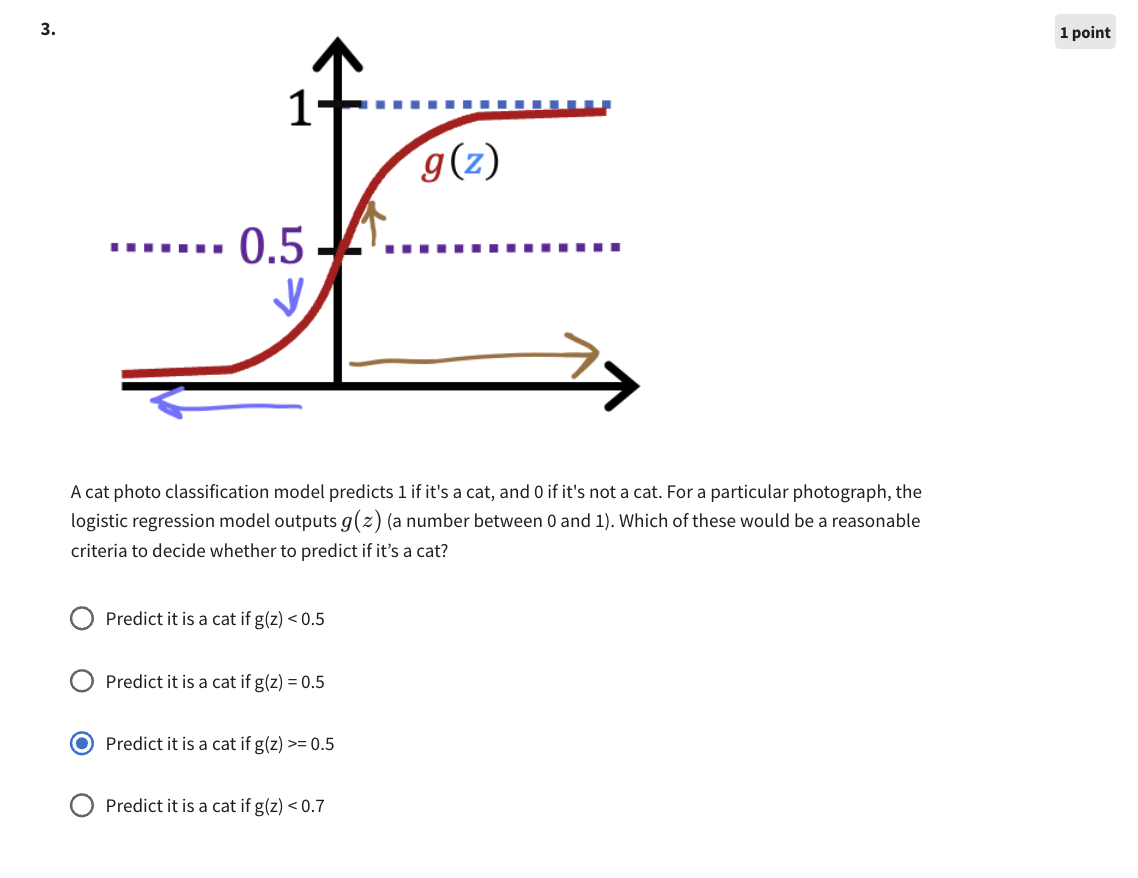
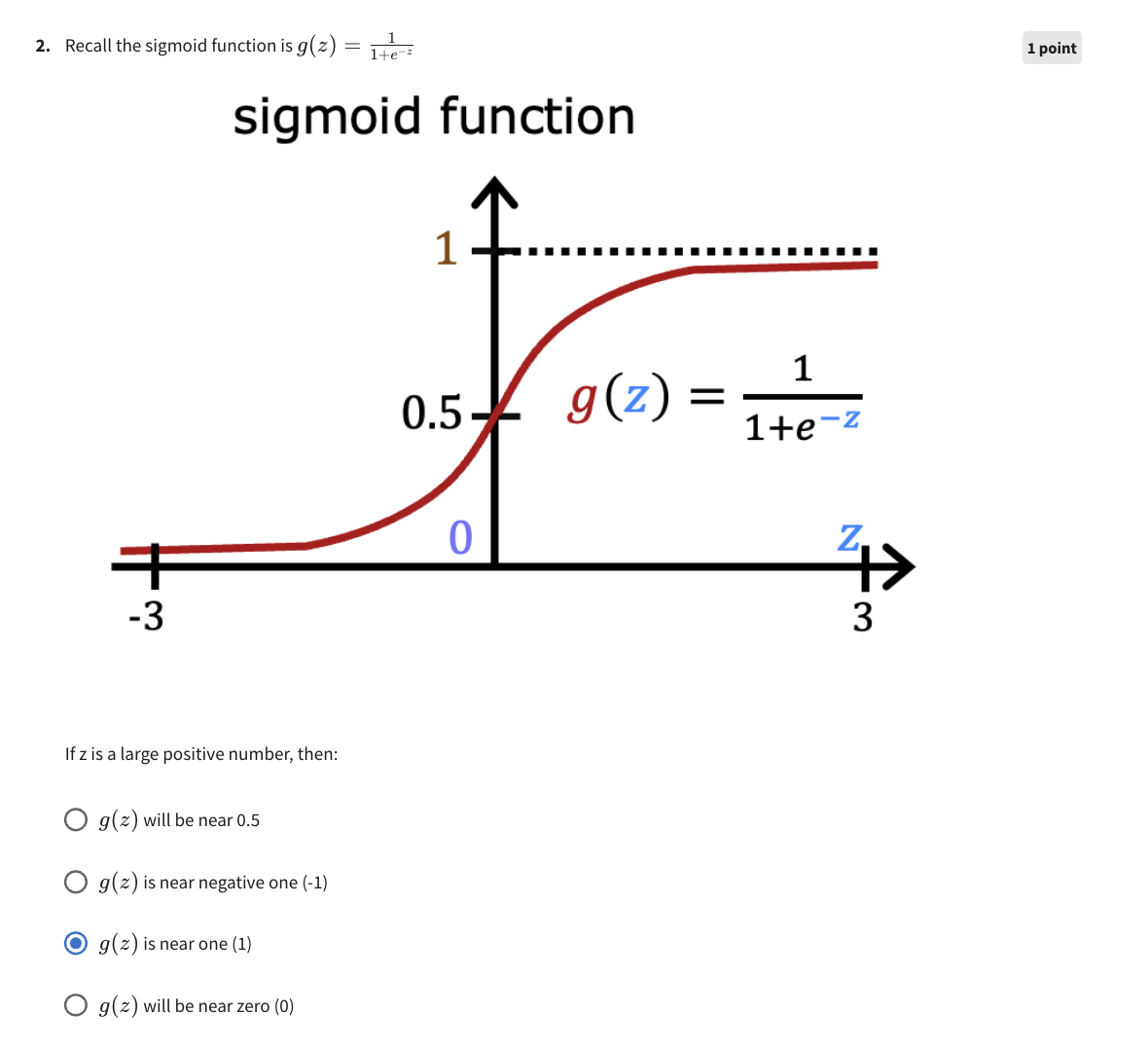
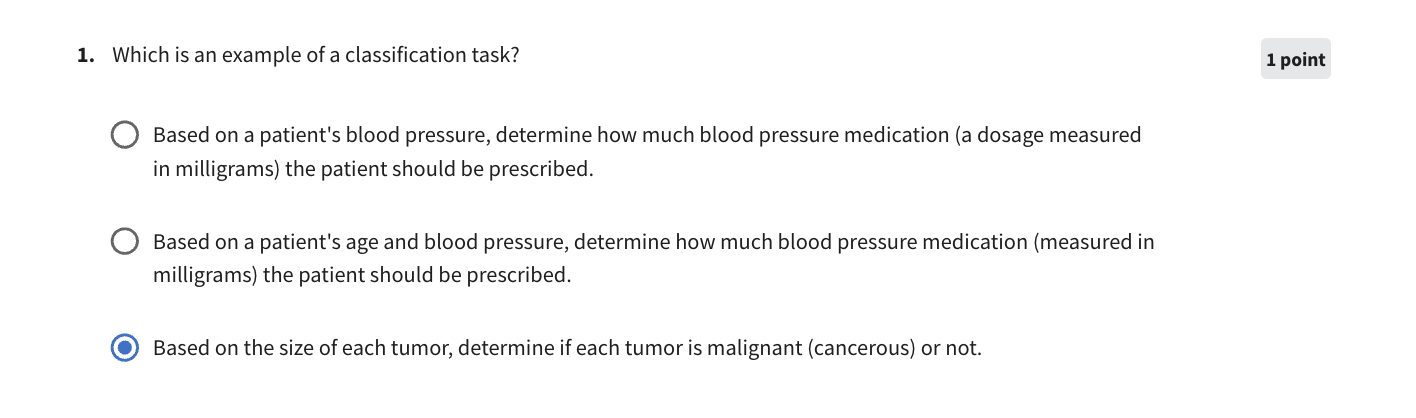
to get decision boundary

→ = 1

You get the following boundary

* + - To get more complex boundaries, you can use higher order polynomial terms

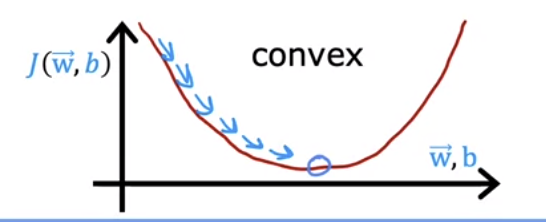
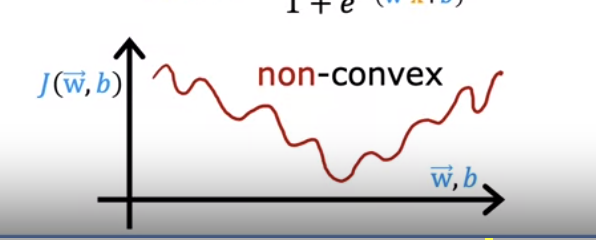
## Practice Quiz: Classification with Logistic Regression



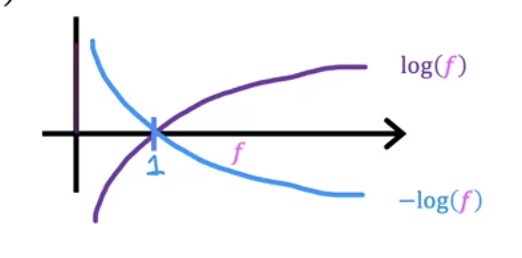
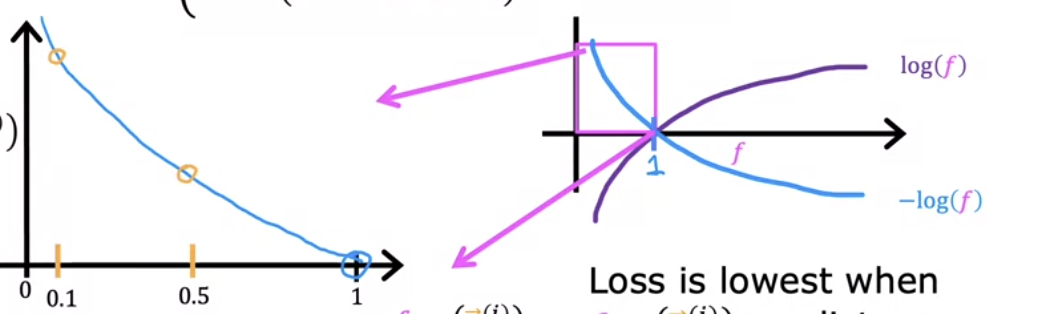
## Cost Function for Logistic Regression

### Cost Function for Logistic Regression

### Example training model

* + For this training set how do we choose the parameters *(w and b)*?
    - Squared Error Cost Function
      * + For linear regression we get a convex/parabola looking cost function
        + This function cannot work for logistic regression 

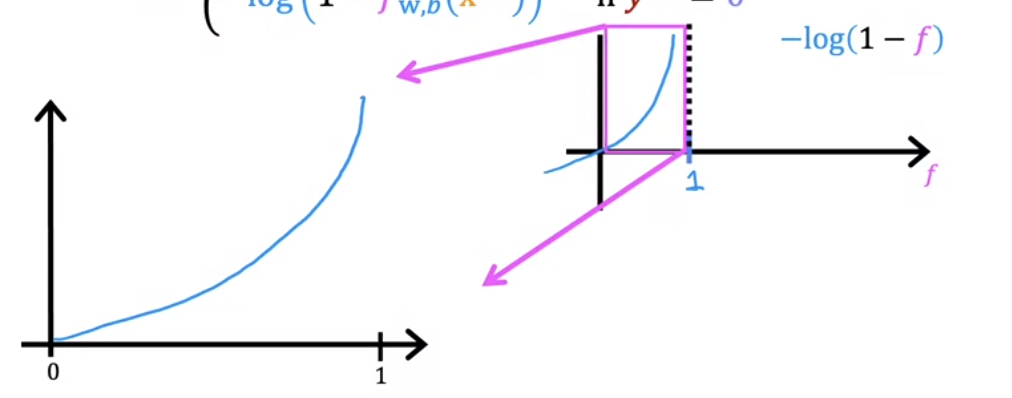
Lots of local minima you can get stuck in

* + - Logistic Loss Function
      * Plotting -log
        + *f* will always be between 0 and 1

First part of loss function

If → 1 then the loss will be 0 since the algorithm is pretty close

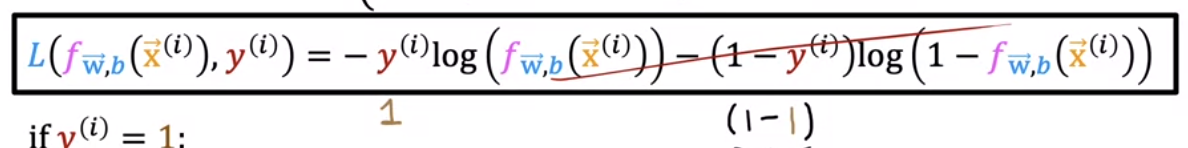
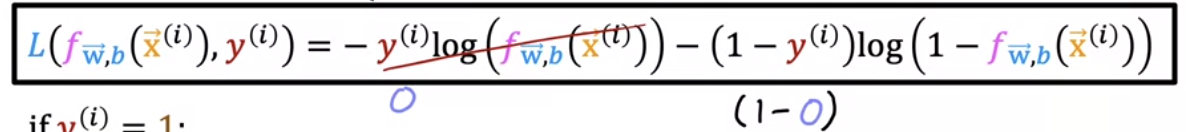
If → 0 then the loss is therefore the algorithm pushes to make more better movements

Second part of the loss function 

If → 0 then you’re very close to optimzing and being close to the true value

If → 1 then the loss will be since you’re further away from the true label 0

### Simplified Cost Function for Logistic Regression

* + - → Could be simplified to
      * If then →
      * If then → 
  + Simplified cost function
    - * → Cost function =
        + → Expanded Notation:

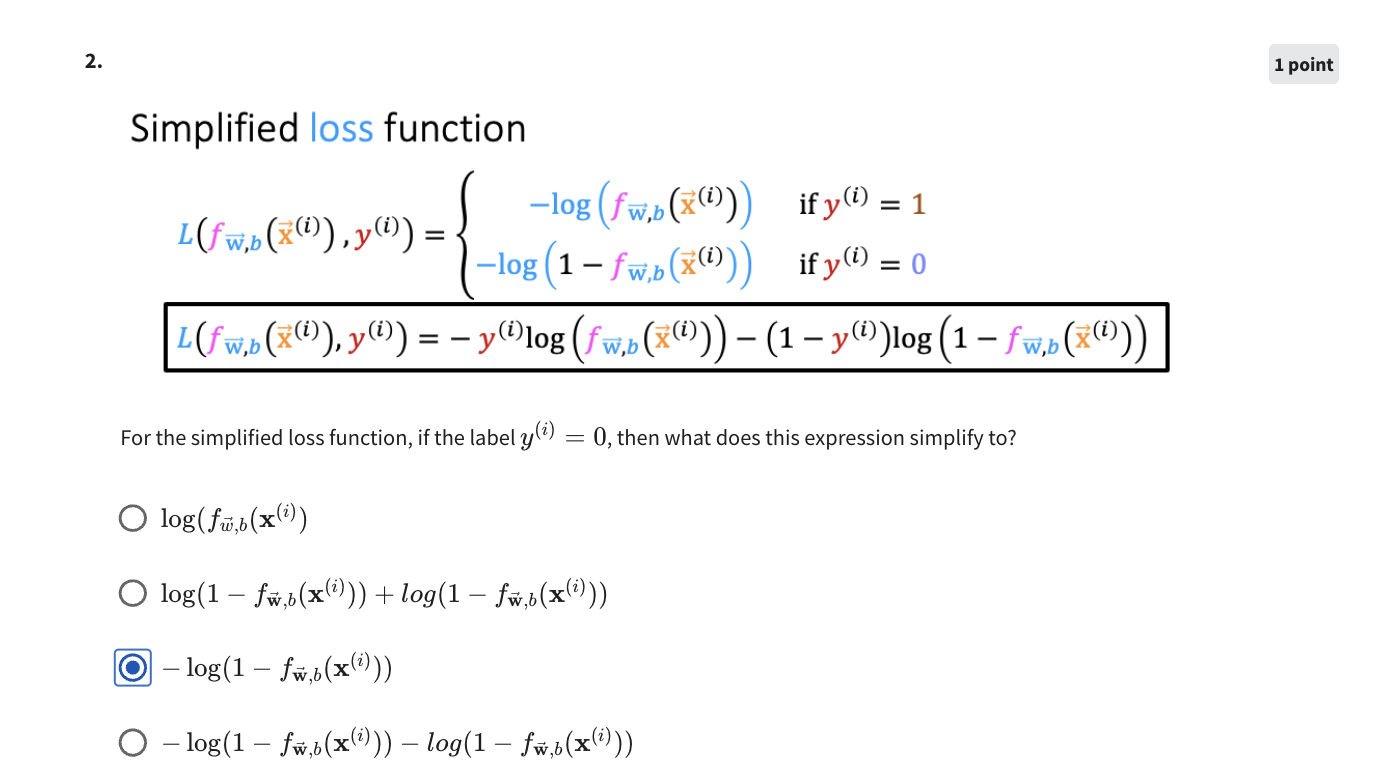
]

## 

## 

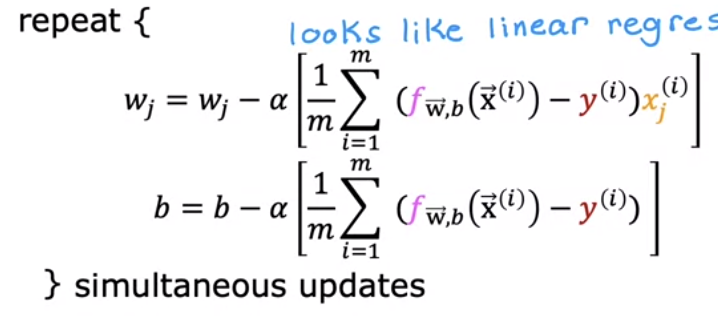
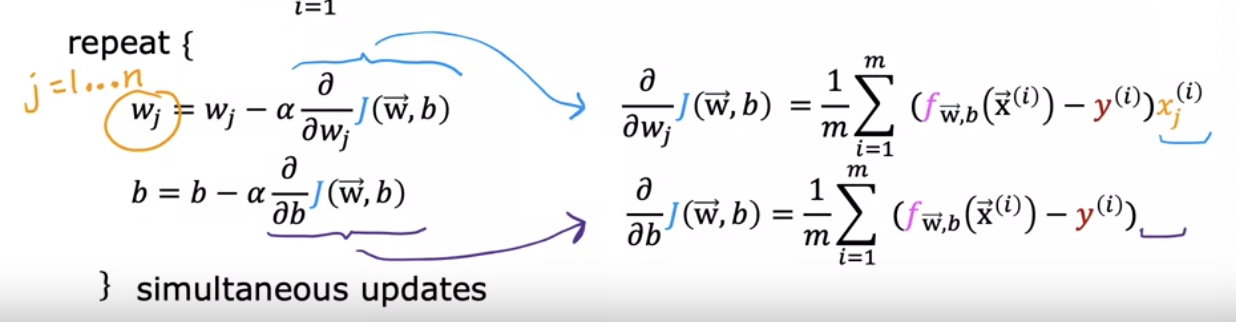
## 

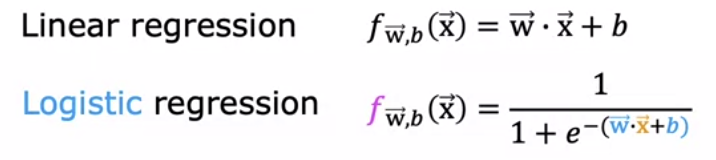
## Practice Quiz: Cost Function for Logistic Regression



## Gradient Descent for Logistic Regression

### Gradient Descent Implementation

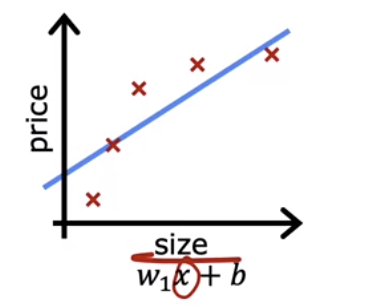
* + Training logistic regression
    - Goal is to find
      * Given a new , output
        + → model can then assess probability that y=1
  + Gradient Descent
    - Cost Function is ]
      * Algorithm would be
        + These equations are the same or very similar to the ones we used in linear regression

This is not the case though since the definition of f(x) is not the same

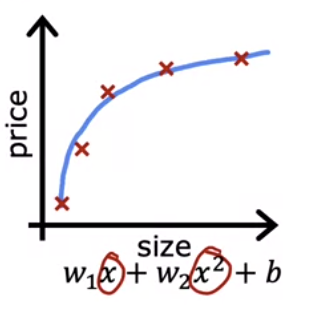
## Practice Quiz: Gradient Descent for Logistic Regression

## The Problem of Overfitting

### The Problem of Overfitting

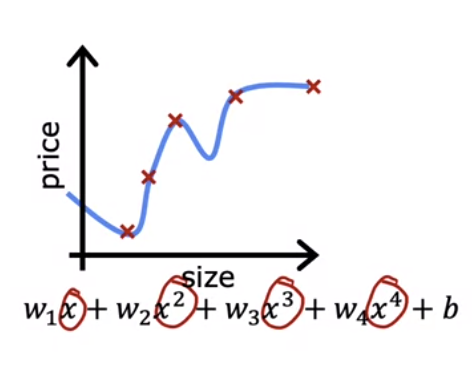
* + What is overfitting and underfitting?
    - Regression Example (House Example)
      * While you could do linear regression, it doesn’t seem like the best fit
        + The model is **underfitting** the training data or has **high bias**

There is a preconception that the housing data will have a linear relationship between the size and the price indicates *high bias*

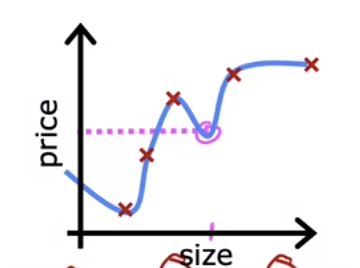
* + - * Fitting a quadratic function
        + Fits the training set pretty well

This is good because it allows for **generalization**

Generalization means the model would be a good predictor of training samples which are not currently in the training set (brand new examples)

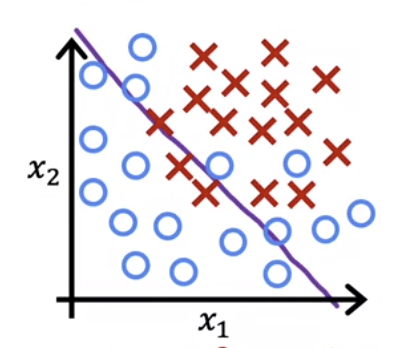
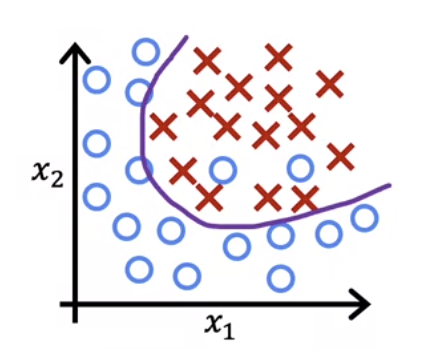
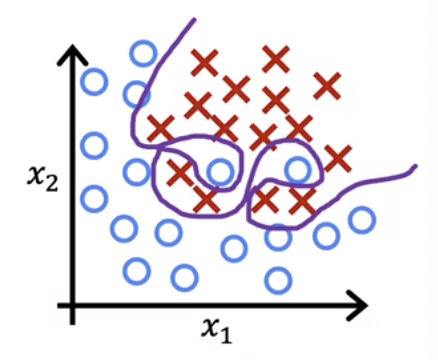
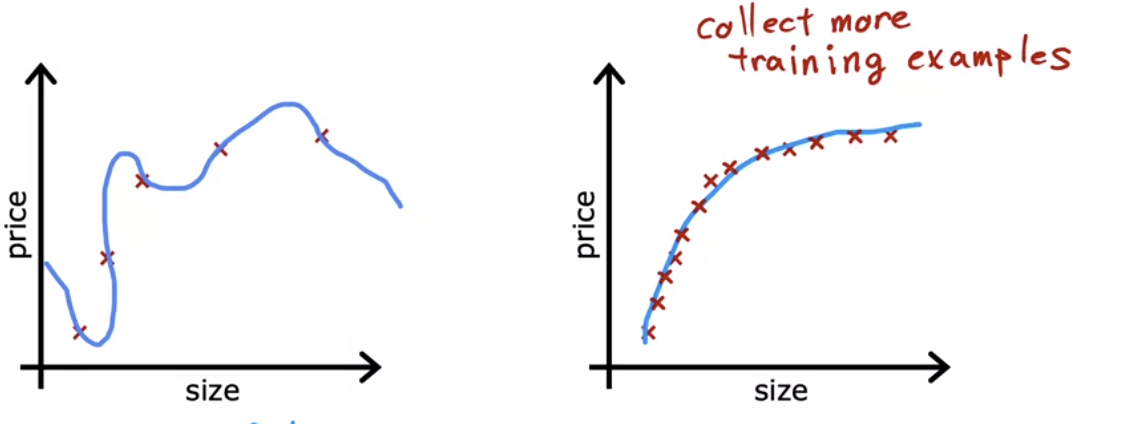
* + - * Polynomial Function
        + Extremely good job at fitting the training data

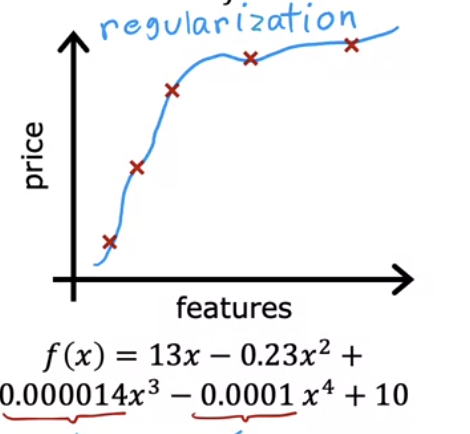
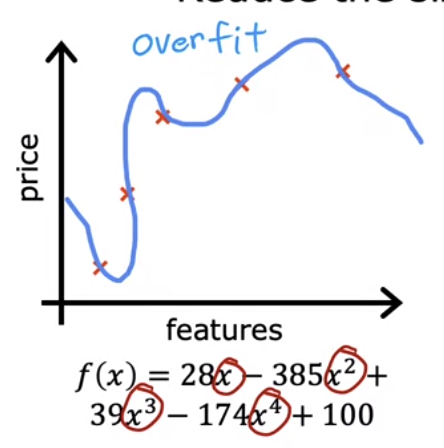
Can get cost function to exactly to 0 since there are no error margins on any training samples

The shape of the model used though **overfits** the data as some results where size is bigger will lead to lower prices

Fits the data *too well* and will not likely generalize to new examples

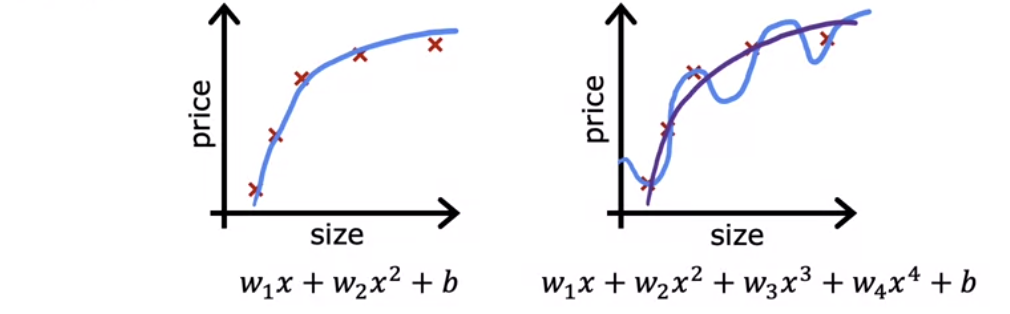
Can also be said to have **high variance** (can use overfitting and high variance interchangeably)

* + Classification
    - Underfitting or high bias
      * While the decision boundary is good, it doesn’t look like the best one therefore the function might be underfitting the data
    - Pretty good fit to the data
    - Overfit to the data 
      * Overly complex division boundary that does not allow for generalization
* Addressing Overfitting
  + Collect more training examples - #1 way to address it
    - More data = function that will likely be less wiggly
  + Select features to include/exclude
    - If you have a lot of features but not enough training data → Overfitting
    - Picking a subset of the most useful features may be helpful in getting to the “just right” point for fitting a model
      * This is called **feature selection**
        + A disadvantage of this is that the algorithm may be throwing away potential useful features
  + Regularization
    - Way to gently reduce the impact of some features without eliminating it outright
      * Shrinks the value of the parameters without making them 0 and eliminating them entirely



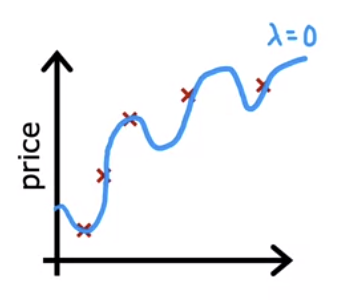
* + - * + Reduces the size of parameters to allow for a better fit to the data

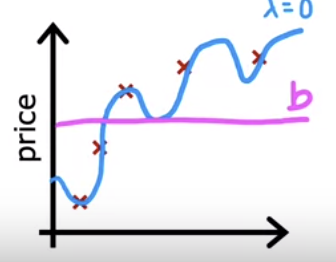
We usually don’t regularize the *b* value

* Cost Function for Regularization
  + Example
    - Suppose you had a way of making *w3* and *w4* really small (close to 0)
      * If you do this you will have a fit more closely to the quadratic function with the tiny contribution from *w3* and *w4*
  + If you have n features
    - You might not know which are the important features and which are not
      * In this case we shrink all the *wj* parameters
    - Example:
      * You have parameters
        + Penalize all them by adding a term to the cost function

→

Lambda is the regularization parameter

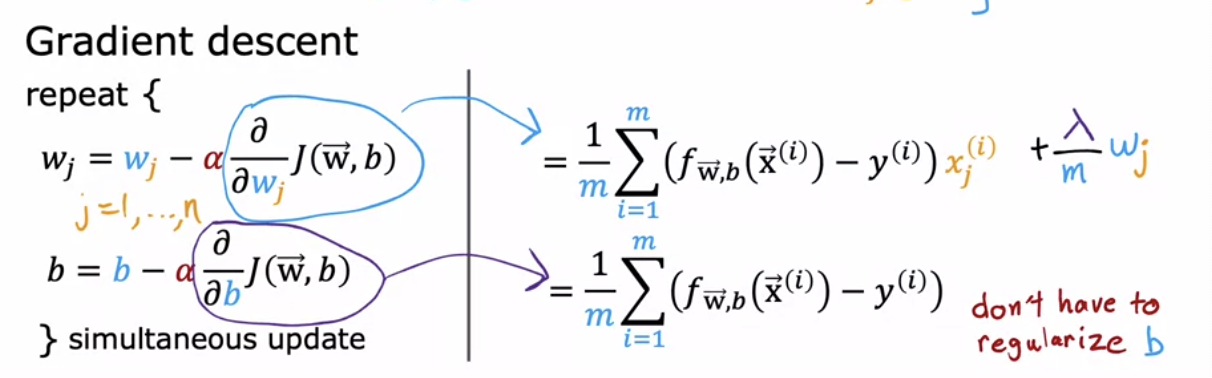
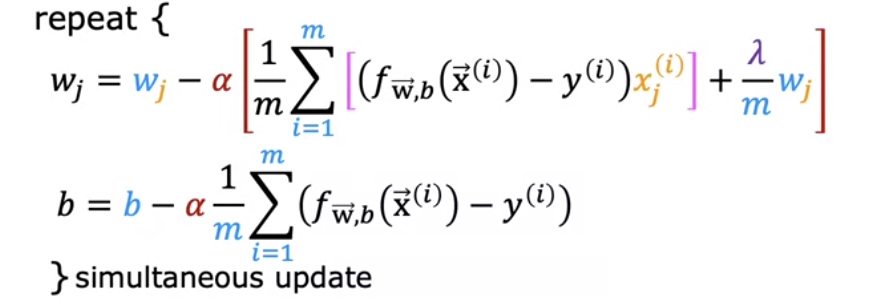
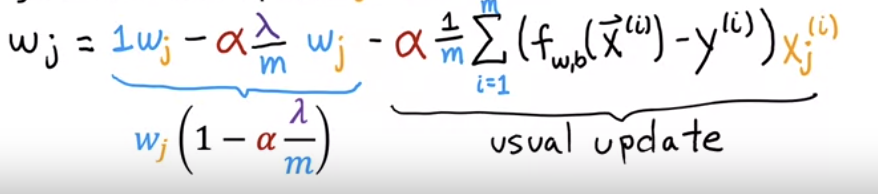
If it is 0, you’re not using the second term → will continue to have a complex model (overfitting)

If it is a big number → placing very heavy weight on the regularization term → will lead all the *wj* terms to be approximately 0 → will lead to → underfitting

Notice both the first and second term are scaled appropriately by 2m

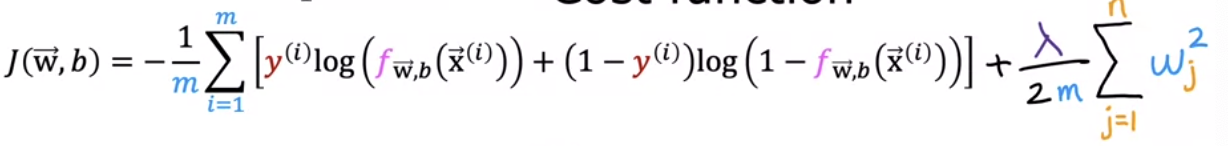
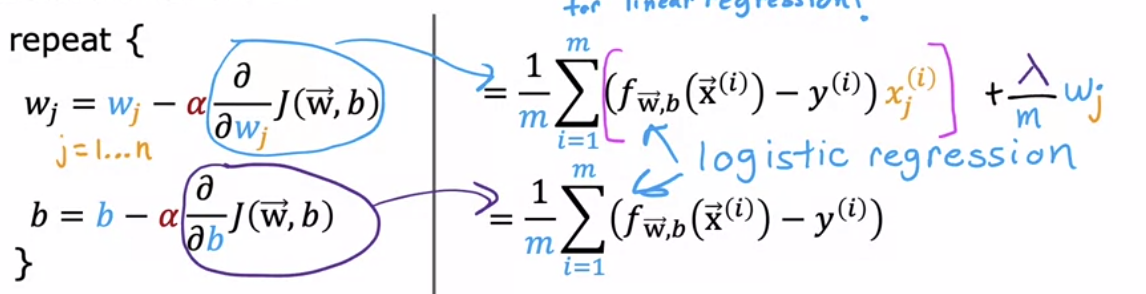
Becomes easier to pick a number for lambda

The first part of the term is referred to as the mean squared error terma and the second part is the regularization term

* Regularized Linear Regression
  + Our Cost function changes a bit since we added the lambda term
    - Implementation
      * Simplifying the first term
        + The second part of the term is the usual update for unregualarized linear regression
        + The first part of the term is usually a number close to 1

This is because are usually small numbers. Divided by a big *m* value (number of examples in the training set), the value that you are multiplying by *wj* is very close to 1

For example if you have the value in the parenthesis = 0.998 → every update the value of *wj* will shrink just a little bit

* Regularized Logistic Regression
  + To add regularization to the cost function, you essentially add the same term to the logistic regression cost function as the linear regression cost function
    - Implementation
      * The only difference is the f(x) function is the sigmoid function applied to z compared to linear regression

## Practice Quiz: The Problem of Overfitting