# Supervised Learning

Tuesday, 8 August 2566 BE 17:17

What is Machine Learning: "Field of study that gives computers the ability to learn without being explicitly programmed"

The more opportunities to learn given to a Machine Learning Program the better

## SUPERVISED LEARNING:

Supervised Learning - Most Used - Rapid Advancements (course 1,2)

Supervised Learning learn from: CORRECT INPUT (X) -->CORRECT OUTPUT LABEL (Y)

Supervised Learning algorithm are given data that INCLUDES the right answers

Right answer: "correct output label (Y) For a INPUT(X)"

**Supervised Learning algorithms LEARN BY SEEING THE RIGHT INPUT & OUTPUTS**

**RESULT OF LEARNING SHOULD BE: INPUT(X) ALONE WITHOUT OUTPUT LABEL (Y) --> RETURN CORRECT PERDICTION**

Example of translation:

INPUT (X) = ENGLISH -> OUTPUT (Y) SPANISH (SHOW 10K CORRECT TRANSLATIONS FOR LEARNING)

FINAL PRODUCT: INPUT(X) = ENGLISH --> OUTPUT (Y) SPANISH PREDICTION

## REGRESSION SUPERVISED LEARNING:

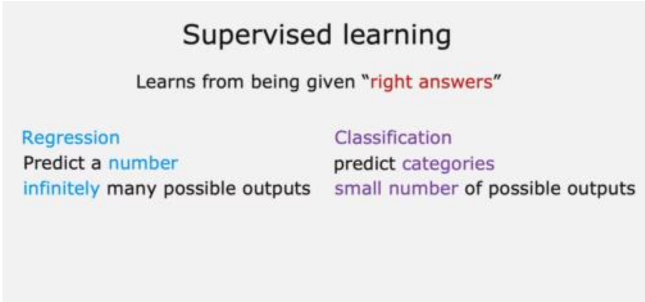
**ANY ML ALGOTHIRM THAT OUTPUTS NUMBERS IS A REGRESSION PROBLEM**

**REGRESSION: FINDING A RELATIONSHIP BETWEEN VARIABLES THEN MAKING PREDICTIONS BASED ON INPUT**

REGRESSION ALGOTHIRM EXAMPLE - USING **LINE** THROUGH THE DATA AND GIVING PREDICTIONS FROM IT WITH ONLY INPUT (X) IN THIS CASE HOUSE SIZE


**THIS IS SUPERVISED LEARNING BECAUSE OF CORRECT EXISTING DATA GIVEN LABEL (Y) GIVEN**

OVERVIEW:




## CLASSIFICATION SUPERVISED LEARNING:

STILL SUPERVISED LEARNING - LEARNS FROM CORRECT ANSWERS - INPUT (X) -> OUTPUT (Y)

**WE ARE TRYING TO PREDICT IF DATA FALLS INTO A SMALL NUMBER OF CATEGORIES GIVEN AN INPUT (SIZE ETC)**

DIFFERENT FROM REGRESSION WHICH TRIES TO PREDICT A NUMBER FROM INIFITE NUMBERS

CLASS & CATEGORY ARE SAME THING IN ML

**CLASSIFICTION ALGOTHIRMS PREDICT WHAT CATEGORIES DATA GOES IN**

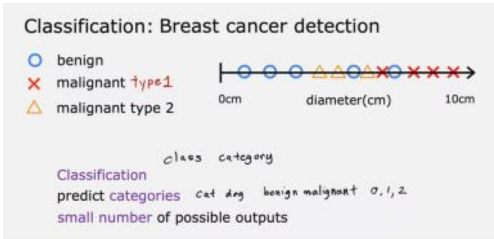
CLASSES DON'T HAVE TO BE NUMBERS - COULD BE PICTURES ETC (IS CAT OR DOG)

CATEGORY 2 - TYPE 2 CANCER - THE TRIANGLE SYMBOL

CATEGORY 1 - FOR MALIGANT CANCER - THE X SYMBOL

CATEGORY 0 - FOR BENIGN CANCER - THE CIRCLE SYMBOL

**THE ML CLASSIFCTION ALGOTHIRM BELOW PREDICTS CANCER TYPE BASED ON SIZE**




THIS IS SUPERVISED LEARNING BECAUSE A SERIES OF CORRECT INPUT (X) AND OUTPUTS (Y) ARE GIVEN THEN PREDICTION IS MADE

**CLASSIFCTION SUPERIVSED ML OFTEN INVLOVES DRAWING BOUNDARIES IN DATASETS FOR PREDICTIONS**

**CLASSIFCATION DOES NOT HAVE MANY OUTPUTS (CAT OR DOG)**

# Unsupervised Learning

Wednesday, 16 August 2566 BE 07:39

**Unsupervised Learning datasets have NO LABELS (Y) ONLY INPUT (X)**

**Unsupervised learning - find something interesting in UNLABELED DATA**

**CLUSTERING ALGOTHIRM - ML MODEL BY ITSELF PLACES UNLABELED DATA INTO CLUSTERS / GROUP**

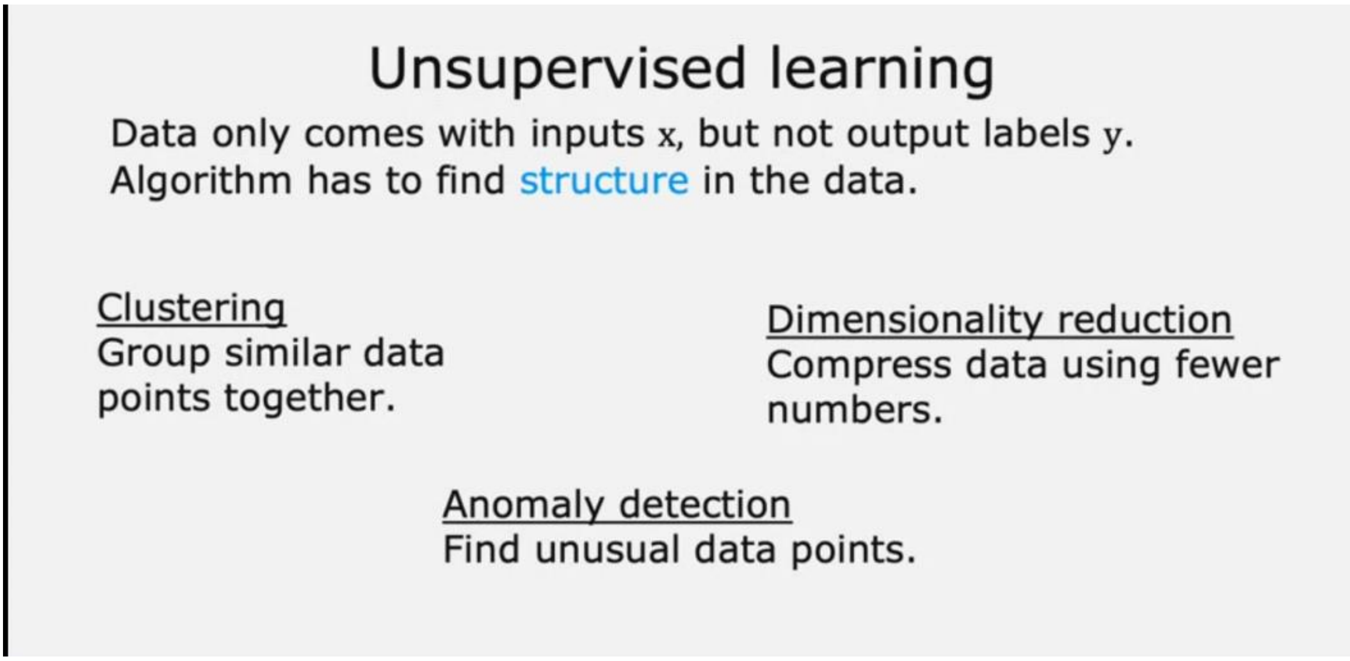
**USES:**

Google News - Look through news stories then CLUSTERS into related news

**UNSUPERIVSED LEARNING CLUSTERING - FINDING STRUCTURE IN DATA BY ITSELF BY MAKING ITS OWN CLUSTERS / GROUPS**

**DIMENSIONALITY REDUCTION - COMPRESSING DATA INTO FEWER NUMBERS**

**ANOMALY DETECTION - OUT OF PATTERN CHANGES - FINDING UNSUAL DATA POINTS**



# Linear Regression With One Variable

Wednesday, 16 August 2566 BE 08:33

**Linear Regression - A straight line relationship through data**

Linear Regression is supervised learning because the dataset contains the "right answers" Input & Correct Ouputs

**Using the correct answers already plotted, the linear Regression algorithm draws a line into the data for its predictions**

### To train this supervised learning model:

Training set including inputs (x) and correct outputs (y) ->

Into learning algorithm (you've made) ->

Function will then be produced (f) CALLED **MODEL**

**FUNCTION TAKES INPUT (X)**

**FUNCTION PRODUCES PREDICTION (Y-HAT // ESTIMATED Y)**

---------------------------------------

**Linear Algorithms are always straight line relationships through a dataset**

**MATHS EXPRESSION:**

$$f_{w,b}(x) = wx + b$$

**ALL THIS MEANS: make a prediction of y (output) using a straight-line function of X (input)**

**W = WEIGHT - SLOPE GRADIENT**

**B = BIAS - Y INTERCEPT**

Why Linear instead of curve etc:

Because its simple & easy to work with, as a foundation for course

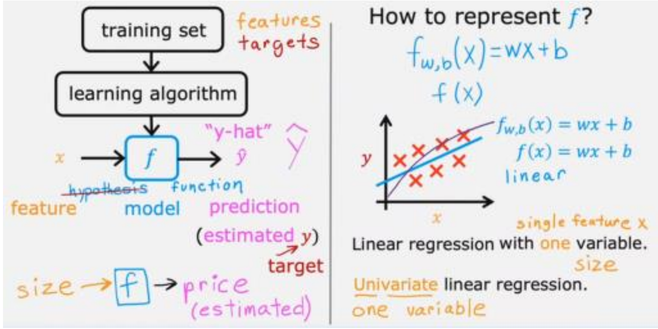
**Linear Regression with one variable means one input**

EVERYTHING EXPLAINED BELOW IN CODE DETAIL:

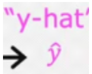
C1_W1_Lab 02_Model...

**t (x)**

Summary Representation:




| General Notation | Description | Python (if applicable) |
|---|---|---|
| $a$ | scalar, non bold | |
| $\mathbf{a}$ | vector, bold | |
| **Regression** | | |
| $\mathbf{x}$ | Training Example feature values (in this lab - Size (1000 sqft)) | `x_train` |
| $\mathbf{y}$ | Training Example targets (in this lab Price (1000s of dollars)) | `y_train` |
| $x^{(i)}$, $y^{(i)}$ | $i_{th}$ Training Example | `x_i`, `y_i` |
| m | Number of training examples | `m` |
| $w$ | parameter: weight | `w` |
| $b$ | parameter: bias | `b` |
| $f_{w,b}(x^{(i)})$ | The result of the model evaluation at $x^{(i)}$ parameterized by $w, b$: $f_{w,b}(x^{(i)}) = wx^{(i)} + b$ | `f_wb` |



# Linear Regression Lab (one var) Explained

Wednesday, 16 August 2566 BE 09:55

In the file below there is a series of notes on how Linear Regression can be done simply

$$f_{w,b}(x) = wx + b$$

The weight (w) and bias (b) are adjusted during training to fit the data as closely as possible

C1_W1_Lab
02_Mode...

# Cost Function Formula

Thursday, 17 August 2566 BE 06:14

$$\text{Model: } f_{w,b}(x) = wx + b$$

**A COST FUNCTION TELLS YOU THE GAP BETWEEN YOUR "PREDICTION" AT ITS CURRENT PARAMTER SETTINGS AND THE CORRECT VALUES**

**w (weight) & b (bias) are parameters - what's adjusted during training**

Changing w (weight) and/or b (bias) will draw a different line on the graph / give different data

**w (weight) gives you the "slope" / gradient of the line - rate it increases / decreases at**

**b (bias) gives you the y-intercept) / where the line starts on the graph**

There are loads of different types of cost function

squared error cost function very popular for regression problems

**The notion "J" is always used to represent cost function**

How to find correct w,b for dataset (SQUARED COST FUNCTION):

**First make a COST FUNCTION to find out what the current "error" is**

$$\left(\underbrace{\hat{y} - y}_{\text{error}}\right)$$

**Take the current predictions output (y-hat)and minus the correct data to find the "error gap"**

Squared Error Cost FUNCTION BREAKDOWN:

SWIGGLY THING IS FOR LOOP

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{error}}\right)^2$$

$m$ = number of training examples

m = amount of data in dataset

"y-hat" = current prediction from model

Y = real value

$$\sum_{i=1}^{m}$$

**Means do this for all data in "m" BASICALLY FOR LOOP**

$$\frac{1}{2m}$$

**Ensures we compute the "average" Error instead of real error - To save time and money**

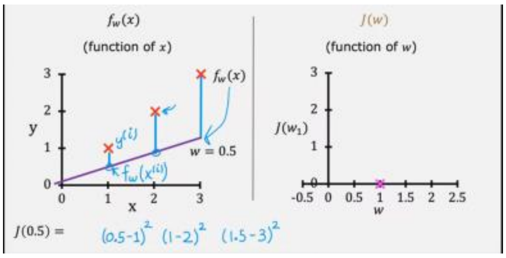
# Cost Function intuition

Thursday, 17 August 2566 BE 06:49

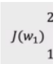
Cost function tells you the difference between your model, at its current parameter settings, and the real data.

Remember w controls slope of the line

**Remember your cost function is giving you the "error" difference between prediction and the real value, not the position**



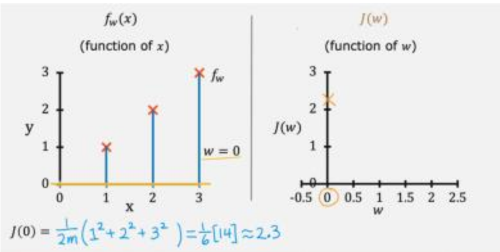

All data for the cost function can be gathered via existing data




**THIS MEANS THE GAP BETWEEN MODEL VS VALUE**

FURTHER EXPLAINED SQUARED COST FUNCTION:

**COST FUNCTION DEPENDS COMPLETELY ON CURRENT MODELS WEIGHTING**




**w (weight) = THE SLOPE OF MODEL**

**YELLOW LINE CURRENT MODEL**

**BLUE MATHS IS COST FUNCTION**

$(1^2+2^2+3^2) =$

**Above the difference between the "yellow line" (OUR PREDICTION) and the real data**

**Averaging to a 2.3 ERROR RATE OUTPUT FROM COST FUNCTION FOR THIS WEIGHTING**

$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\Big(\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{error}}\Big)^2$$

m = number of training examples

**Y-hat is your current weightings model predictions and "y" is the real value - thus the bit in the brackets gives the difference FINALLY PRODUCING AVERAGE FOR ALL**

**Remember Y is always output and that's what the cost function is checking the "error rate" for**

Explainer note:

OUTPUT OF THIS: THE AVERAGE DIFFERENCE BETWEEN THE MODEL AND VALUES FOR ENTIRE DATASET

If W was made 0.5 or 1 or whatever then the input to cost function would be different - so its output would also be different

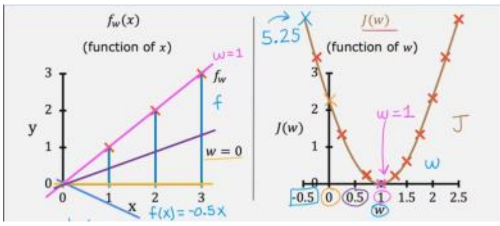
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\Big(\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{error}}\Big)^2$$

m = number of training examples

THEN YOU KEEP "TRAINING" BY ADJUSTING WEIGHTING AND BIAS UNTIL SMALLEST NUMBER

**For each value produced by a cost function (when the weighting / w is different) represents a different line through the data.**

As you can see




**For this model the correct weighting according to squared cost function is 1 (right hand graph)**

**Left hand graph confirms this**

**COST FUNCTIONS MEASURE HOW ACCURATE YOUR REGRESSION MODEL IS**

**Always pick smallest cost function output**

THESE EXAMPLES ARE MAINLY JUST WEIGHTING

REMEMBER B IS Y INTERCEPT

# Visualizing The Cost Function (with bias)

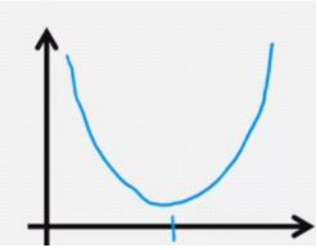
Thursday, 17 August 2566 BE 08:15

**Cost functions will normally be automated via things like gradient decent**

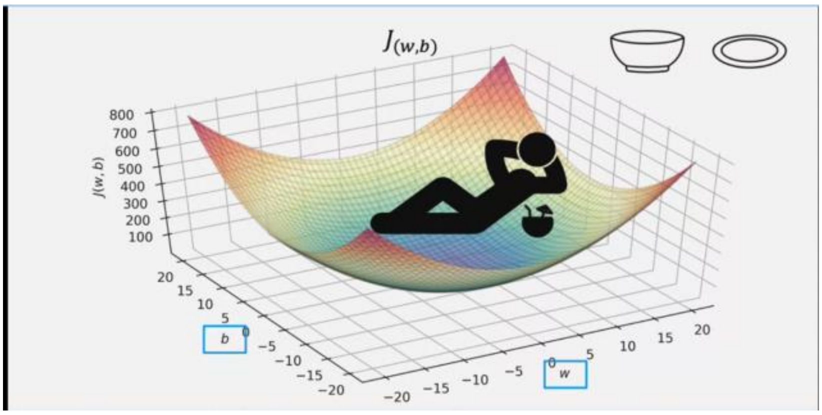
**Weighting (w) = gradient / slope**

**Bias (b) = y intercept**

**Both have a strong U shape when put on a line graph like:**



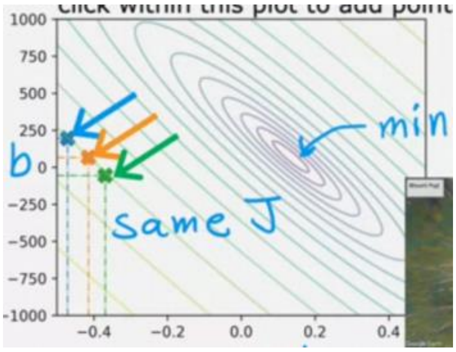
**The Two "U" from W & B make a bowl:**



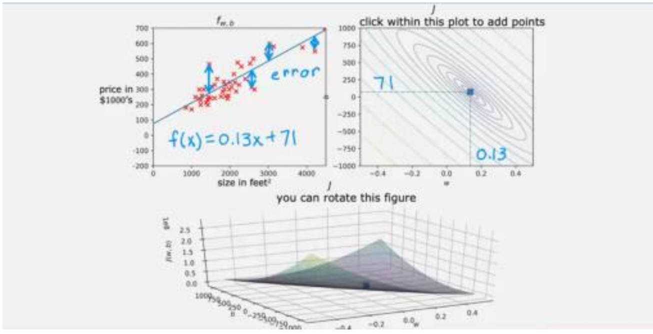

**Each point on this is a different bias and weight**

By "slicing" the plot you get a CONTOUR plot which is all the Weights and Bias's that give you the same average cost output

The lowest point on the bowl is the best one OR The center of the contour like:




Graphs Explained:




LEFT HAND GRAPH:

MODEL - WITH CURRENT BIAS AND WEIGHT

RIGHT HAND GRAPH:

COST FUNCTION OUTPUT - ITS SMALL SO ITS ALMOST PERFECT

# Squared Cost Function Lab

Monday, 21 August 2566 BE 06:17

C1_W1_Lab
03_Cost_f...

Find Above a explained version of the Squared Cost Function including using the Matplotlib - a really useful visualization lib

# Gradient Descent

Monday, 21 August 2566 BE 06:20

**The point of gradient descent is to find the lowest output of the cost function**

**Gradient Descent finds the smallest possible "average cost" / Minimizes average cost (w,b)**

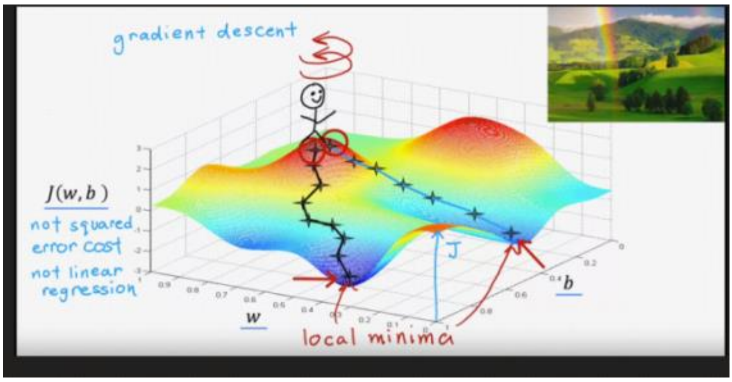
**Works with any cost function no matter how many parameters (w1,b1,w2,b2 etc)**

Start with a few guesses (weight=0, bias=0)

**IF Not Squared Error Cost You may not get a "U" SHAPE**

**Gradient Decent takes small steps in the path of "steepest decent" until it reaches lowest point**

**Depending on where you start you may get a different "local minima" / lowest point**




Convergence is when you find the REAL lowest point / best params not just local

# Implementing Gradient Descent

Monday, 21 August 2566 BE 06:50

Below is the Gradient Descent algorithm - for finding the best weight & bias

"=" in this document means assignment not comparison e.g. w is assigned the result

Gradient Descent for W (weight):

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

"=" in this case, is an assignment to W (stored in w)

$$\alpha$$

**"Alpha" (above) is the learning rate - Controls how big a step we take down hill (descend) -** This is normally a small positive number, the bigger the number, the bigger the step.

**"Alpha" (above) Controls how big an update is made to Weight & Bias each time Gradient Descent is run**

$$\frac{\partial}{\partial w} J(w,b)$$

**Derivative of our cost function - Controls what direction we take our step in and our learning rate a bit - go to next note for more info**

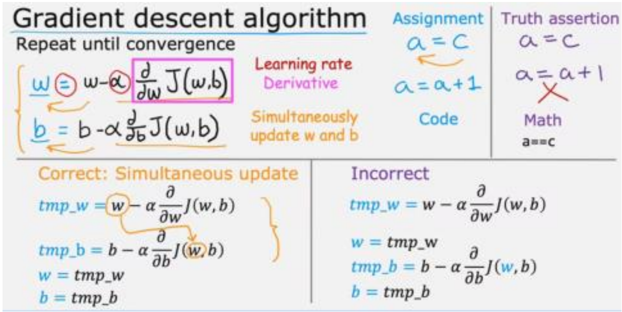
Gradient Descent for B (Bias):

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b)$$

**The Goal is to reach "convergence" when the results from gradient descent don't change much**

**W AND B SHOULD BE UPDATED AT SAME TIME FOR GRADIENT DESCENT - SO THEY DON'T GET OUT OF SYNC**

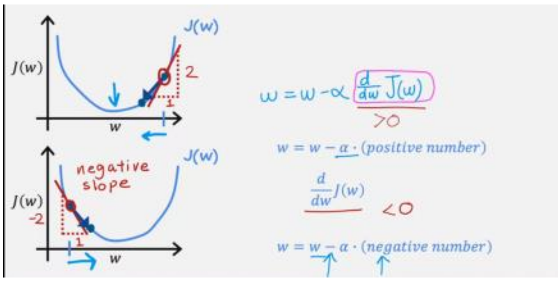
W & B MUST BE UPDATED AT SAME TIME - LOOK AT IMAGE BELOW:



# Gradient Descent Intuition

Monday, 21 August 2566 BE 07:27

**The derivative works by drawing tangent lines**




**Always calculate the derivative first when programming gradient descent**

**The derivative gives the slope of the tangent at the parameter (W's) current location - Controlling it movement**

**If output is positive go left (DECREASE OVERALL)**

**If output is negative go right (INCREASE OVERALL)**

This is done until you reach local minima

## The reason the derivative controls direction:

**Because this is the derivative is the slope of the tangent**

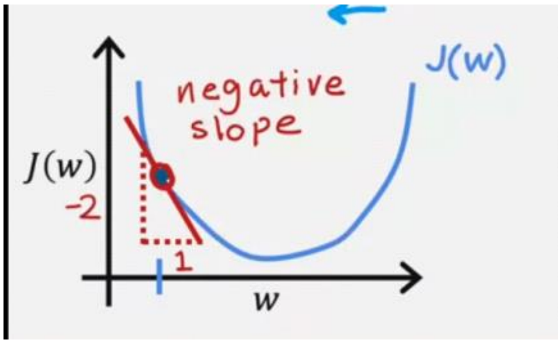

**2 over 1 = 1 positive WHICH MEANS**

$$w = w - \alpha \cdot (positive\ number)$$

**This leads to decreasing W (because minus overall)**

**Learning rate is always positive number**

## Negative example - Gradient Descent:




**-2 / 1 = -2 NEGATIVE**

$$w = w - \alpha \cdot (negative\ number)$$

**The leads to increasing W (because positive overall)**

**N**

$$\frac{\partial}{\partial w} J(w,b)$$

**THE DERIVATIVE RESULT OF THE CURRENT PARAMETERS TANGET SLOPE**

**THE STEEPER THE PARAMETER SLOPE , THE BIGGER THE STEP**

# Learning Rate

Monday, 21 August 2566 BE 08:18

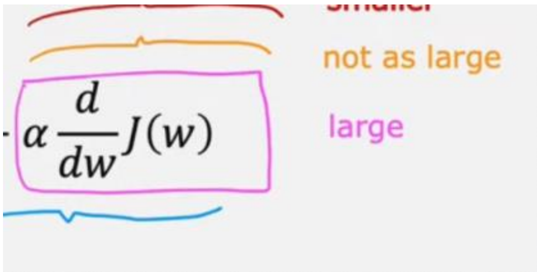
**Learning rate - size of parameter changes / size of descent**

**Learning rate too small - It will work, but it will take ages to make the amount of steps**

**Learning rate too large - you could skip over the minimum**

If learning rate is too large - Overshoot / fail to converge

### What happens as you get closer to minimum:

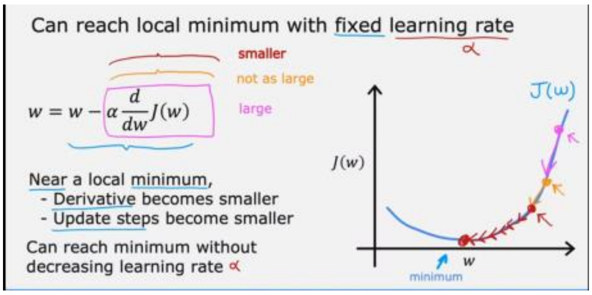



**Because the derivative tangent is smaller (above) smaller steps are made**

**Parameter updates become smaller**

**YOU CAN REACH MINIMUM WITHOUT DECREASING LEARNIG RATE (a)**

**As the gradient of the graph decreases the derivative shrinks, decreasing learning**




**Because the number the deviation is producing is getting smaller**

# Gradient Descent for linear regression (one var)

Monday, 21 August 2566 BE 08:44

## Gradient Descent For Linear Regression Explained:

**Always calculate gradient / derivative first**

In lecture, *gradient descent* was described as:

$$\begin{align*} &\text{repeat until convergence: } \lbrace \\ &\quad w = w - \alpha \frac{\partial J(w,b)}{\partial w} \\ &\quad b = b - \alpha \frac{\partial J(w,b)}{\partial b} \\ &\rbrace \end{align*}$$

where, parameters $w$, $b$ are updated simultaneously.
The gradient is defined as:

$$\begin{align} \frac{\partial J(w,b)}{\partial w} &= \frac{1}{m} \sum\limits_{i = 0}^{m-1} (f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)} \\ \frac{\partial J(w,b)}{\partial b} &= \frac{1}{m} \sum\limits_{i = 0}^{m-1} (f_{w,b}(x^{(i)}) - y^{(i)}) \end{align}$$

**REMINDER: UPDATE SIMULTANEOUSLY**

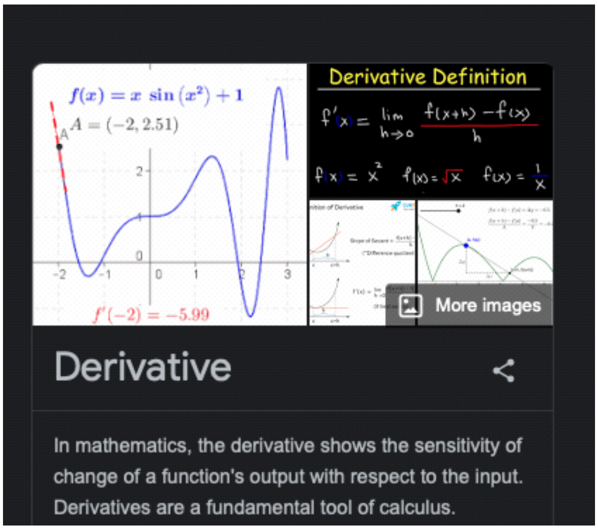
**SQUARED COST FUNCTION FOR LINEAR REGRESSION NEVER HAS MORE THAN ONE MINIMUM**

**Convex function never has more than one minimum (bowl shape) for linear regression**

The local minimum is always, the convergence point in linear regression

# Calculating Derivative Terms

Monday, 21 August 2566 BE 08:47




About featured snippets • Feedback

## Partial derivative

In mathematics, a partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant. Partial derivatives are used in vector calculus and differential geometry. Wikipedia

A Partial Derivative measures how sensitive a function is to one its parameters changing

Derivate measures how sensitive a function is to an input change

I don't understand this much but I'll save this anyways

**WEIGHT**

(Optional)

$$\frac{\partial}{\partial w}J(w,b) = \frac{d}{dw}\frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)})-y^{(i)}\right)^2 = \frac{d}{dw}\frac{1}{2m}\sum_{i=1}^{m}\left(wx^{(i)}+b-y^{(i)}\right)^2$$

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

**BIAS**

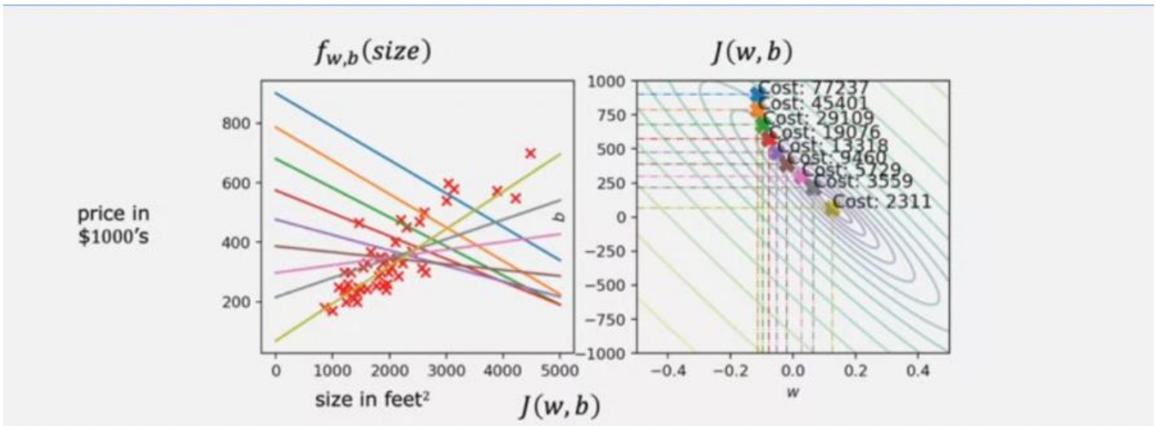
$$\frac{\partial}{\partial b}J(w,b) = \frac{d}{db}\frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)})-y^{(i)}\right)^2 = \frac{d}{db}\frac{1}{2m}\sum_{i=1}^{m}\left(wx^{(i)}+b-y^{(i)}\right)^2$$

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

no $x^{(i)}$

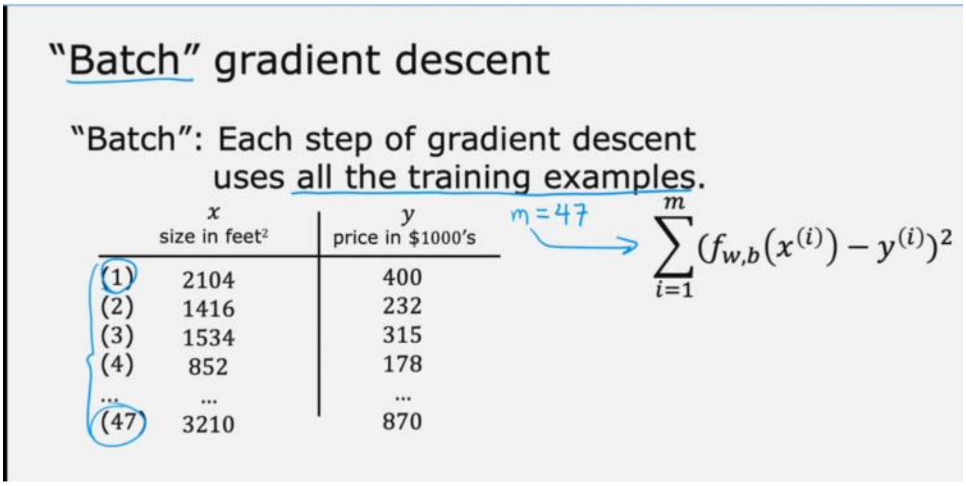
# Running Gradient Descent

Monday, 21 August 2566 BE 09:30




Shows how the model changes when gradient descent is used

**This is batch Gradient Descent because using a cost function with all the training data considered**



# Batch Gradient Descent Lab

Monday, 28 August 2566 BE 06:33

C1_W1_Lab
04_Gradi...

Personally noted version

# Multiple features

Monday, 28 August 2566 BE 06:34

Linear Regression with multiple variables / inputs is called "multiple linear regression"

Vectorization is about putting all these variables "inputs" into groups

Features are your "inputs"

**$X_1$, $X_2$ are used to show different variables**

$x_j = j^{th}$ feature

Used to show the different features (inputs)

$\vec{x}^{(i)}$ = features of $i^{th}$ training example

**Gives you specific x (as before) in multiple variable, this becomes a vector (a group of inputs) like:**

$$\vec{x}^{(2)} = [1416 \quad 3 \quad 2 \quad 40]$$

**ABOVE IS A VECTOR OF INPUTS**

$\vec{x}$

**THIS IS MUTLIPLE FEATURES / A VECTOR**

**To get a specific feature "x" NOT GROUP do:**

$$x_3^{(2)}$$

This means: 2nd vector, 3rd feature // left corner is vector (row) and bottom left is where on that row

Linear Regression Multiple Features Model:

$$f_{w,b}(x) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

Standard format:

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

**ARROWS SHOW ROW VECTORS / LISTS**

**The "dot" means Times W & X IN PAIRS THEN SUM ENTIRE LIST**

**"Dot product" times together in pairs "$W_1$ TIMES $X_1$, $W_2$ TIMES $X_2$ ETC THEN SUM**

Each Feature (x) has its own "weighting", $X_3$ Will have weighting / gradient $W_3$

This means cost functions + gradient descent will need to be done on each one

X **Vectors often contain training data during training (feature) e.g**

$x_j^{(i)}$

**Otherwise they are inputs**

# Vectorization

Monday, 28 August 2566 BE 07:18

**Having data in vectors allows you to do calculations on mass, in parallel, useful on large datasets**

**Each new feature / "input" needs its own weighting, so put both into vectors**

**Linear Regression with Vectorization:**

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

"W arrow", X "X arrow" means vectors (arrays basically)

Using Vectors in python:

```
f = np.dot(w,x) + b
```

**Dot will go through both arrays - add times each pair then sum and add bias**

Basically doing this:

```
for j in range(0,n):
    f = f + w[j] * x[j]
f = f + b
```

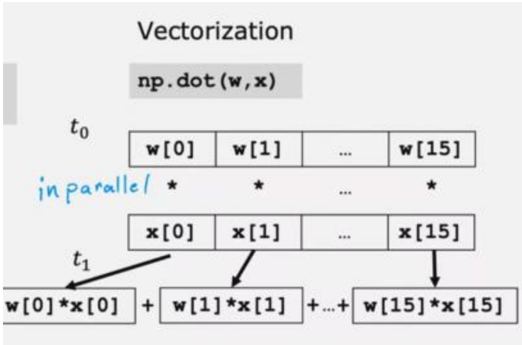
But with much more efficiency and cleaner code

**Benefits of vectorization:**

Faster and more efficient code - **Uses Hardware including GPU "Parallel hardware"**

Cleaner code

Vectorization Python Explained for Multiple Linear Regression Algo:




**Vectorization allows the computer to multi-task**

**Vectorization allows adding all pairs at once**

**Vectorization then allows for the summing of all the outputs at once**

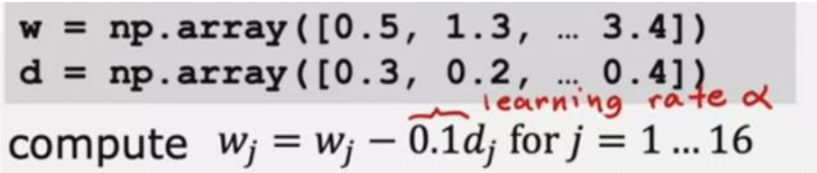
# Basic Multiple Gradient Descent with vectorization note (check next note for more detail)

Monday, 28 August 2566 BE 07:43

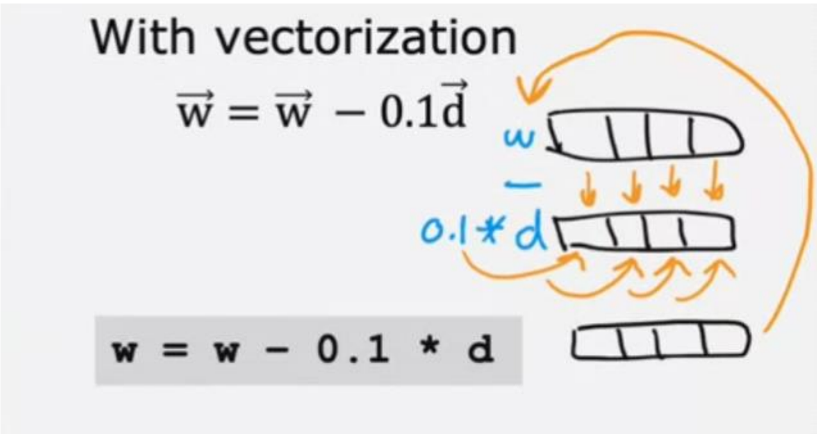
**Because you have serval of inputs, you have serval weightings (w), which means serval derivatives for gradient descent**

**ALL OF THESE SHOULD BE IN VECTORS FOR np.dot() USAGE**

**BIAS IS IGNORED HERE**




**Vector of derivatives / current tangent for each WEIGHTING (w) because each of our features needs its own (first note check for more detail)**




**The Weight is calculated for each derivative all at once because of vectorization dot function**

**Learning rate: 0.1**

**Vectorization allows us to do gradient descent all at once**

**Because both w (weight) and d (derivatives) are both in vectors, the above calculate is done in parallel**

# Python, NumPy and Vectorization

Monday, 28 August 2566 BE 08:19

C1_W2_Lab 01_Pytho...

C1_W2_Lab 01_Pytho...

C1_W2_Lab 01_Pytho...

Remember that for vectorization pairs, arrays must be same shape (including np.dot() )

C1_W2_Lab 02_Multi...

C1_W2_Lab 02_Multi...

UPDATED VERSION

# Gradient descent for multiple linear regression

Monday, 28 August 2566 BE 08:37

| | Previous notation | Vector notation |
|---|---|---|
| Parameters | $w_1, \cdots, w_n$<br>$b$ | vector of length n<br>$\vec{w} = [w_1 \cdots w_n]$<br>$b$ still a number |
| Model | $f_{\vec{w},b}(\vec{x}) = w_1x_1 + \cdots + w_nx_n + b$ | $f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$<br>dot product |
| Cost function | $J(\underbrace{w_1, \cdots, w_n}, b)$ | $J(\vec{w}, b)$ |

Gradient descent

Previous notation:

$$\text{repeat} \{$$
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\underbrace{w_1, \cdots, w_n}, b)$$
$$b = b - \alpha \frac{\partial}{\partial b} J(\underbrace{w_1, \cdots, w_n}, b)$$
$$\}$$

Vector notation:

$$\text{repeat} \{$$
$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$
$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$
$$\}$$

**Because we now have multiple features, "w" / weighting must become a vector for each new "x" / feature**

Vectored Multiple Gradient Descent For Linear Regression:

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right) x_n^{(i)}$$

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

**Updated at same time to stay in-sync**

# `Feature Scaling Part 1

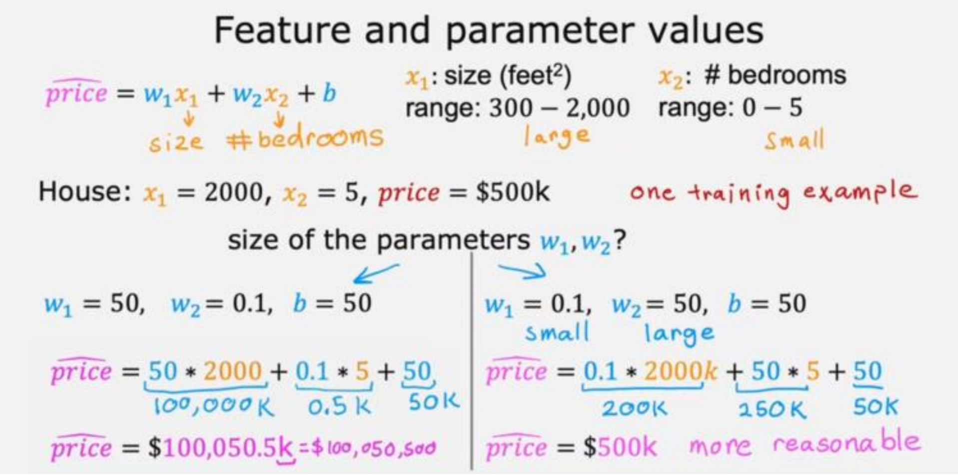
Tuesday, 29 August 2566 BE 06:15

**You must do feature scaling when one feature is much larger than another**

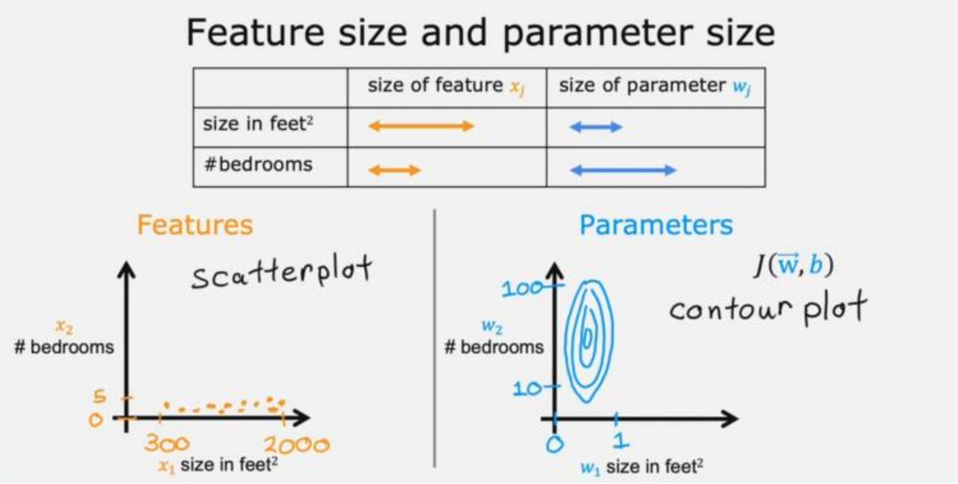
**Feature = input $x_2$ or whatever**

**The bigger the feature (input), the smaller its parameters (weight etc) will be in a good model**

**The smaller the feature, the bigger its parameters will be (weight etc) will be in a good model**




Larger parameters in a cost function are far more sensitive - because a small change can make a big different to results

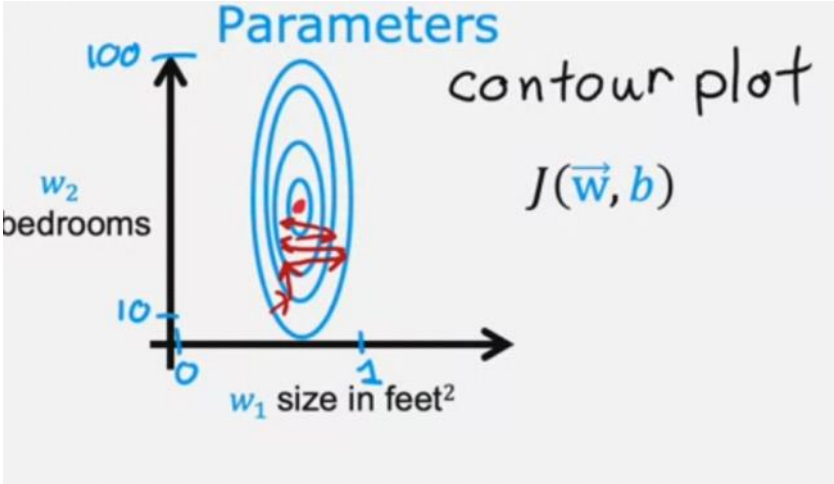



**The contour plot for cost function is far shorter for w1, because it is larger and more sensitive to changes**

## Feature Size and Gradient Descent:

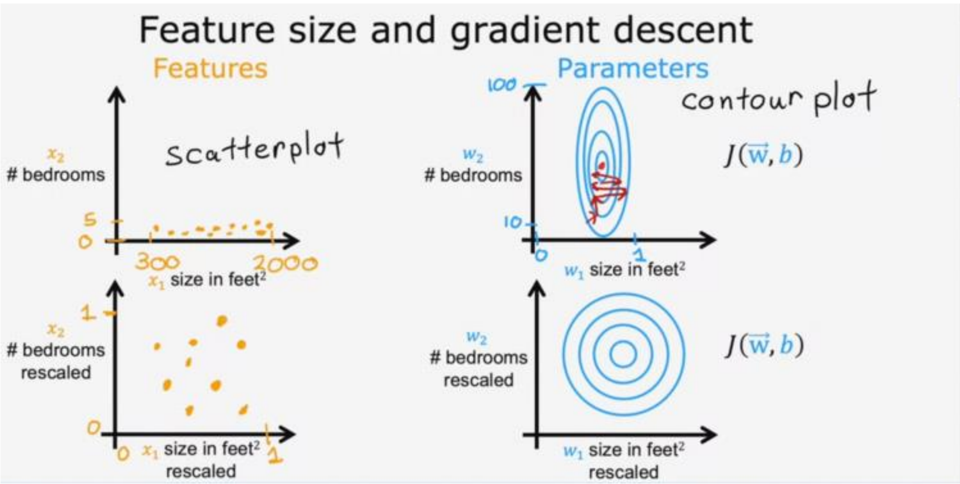
**When you have features (inputs) that take on very different values, gradient descent is slow without scaling**

**If contour plot is "skinny" gradient descent will bounce backward and forward, making it slow**




Feature Scaling:

**By scaling the features to be similar to each other you can get a more "normal" contour plot**



# Feature Scaling Part 2

Tuesday, 29 August 2566 BE 06:47

Applied to cost function

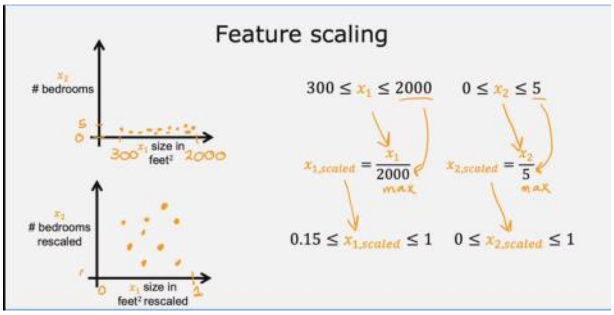
If used, Feature scaling should be run on every feature, not same scale, just same strategy

**You must do feature scaling when one feature is much larger than another**

**To get the correct feature scale:**

**Divide all by the maximum in the range**

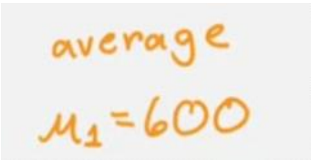
**Do feature scaling individually on each feature (input)**



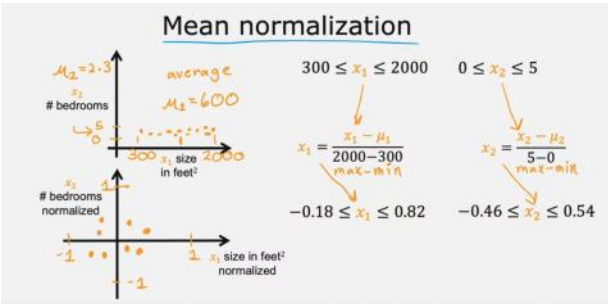

Further scaling, Mean Normalization:

Make all the data centered around 0

**First find the mean for each feature shown as:**




Explained below:




Find the average

Take all values away from the average

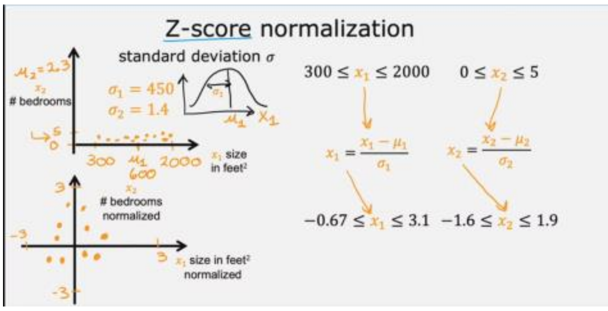
Denominator - take largest dataset member from smallest dataset member

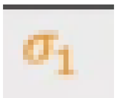
Z-score normalization:

**Calculate standard deviation of each feature**

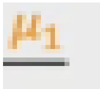
**Find average of each feature**

Example of formula below:




**This means standard deviation / sigma**




**This is the mean (average)**

## Goal of feature scaling:

**Get a good, easy to work with feature range, single digit**

**To ensure all features train at a similar rate and all features are set correctly**

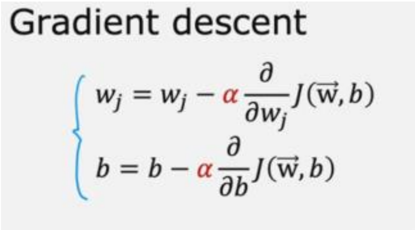
**Allows you to run at much higher learning rate (alpha)**

# Checking Gradient Descent for Convergence

Tuesday, 29 August 2566 BE 07:22

Basically, has gradient descent found the best score from the cost function by modifying parameters or just the local minima

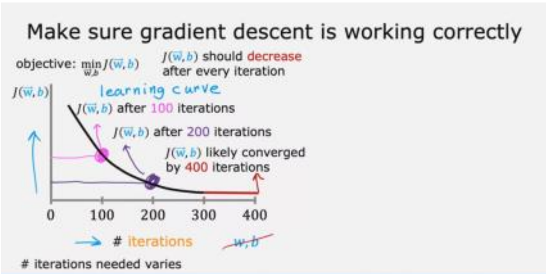
Gradient Descent Reminder:




## Make sure gradient descent is working:

Using a learning curve:

**Below graph is called a learning curve**




**This curve measures the "steps" of gradient descent each simultaneous update of parameters (weight + bias)**

**IF THIS EVER INCREASES, ALPHA / LEARNING RATE IS TOO BIG OR CODE IS BAD**

This graph also helps you know when you can stop training and you've reach convergence

If graph goes flat for extended period, you have reached convergence likely

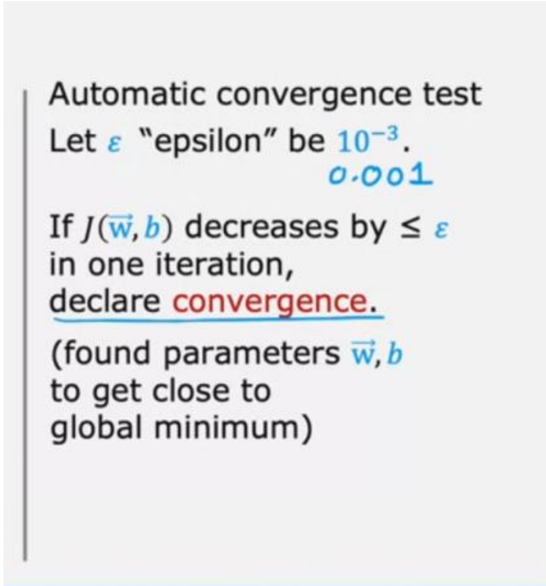
Its almost impossible to know how many iterations a application will need to converge

## Automatic Convergence Test:

**Decide on a very small number (0.001)**

**If learning curve decreases by less than this:**

**Declare convergence**




**Graphs are most of the time, better**

**Because its hard to pick the right number**

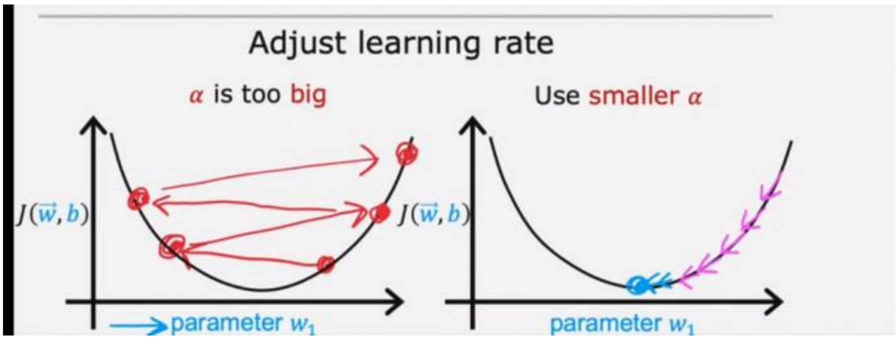
# Choosing The Learning Rate

Tuesday, 29 August 2566 BE 07:52

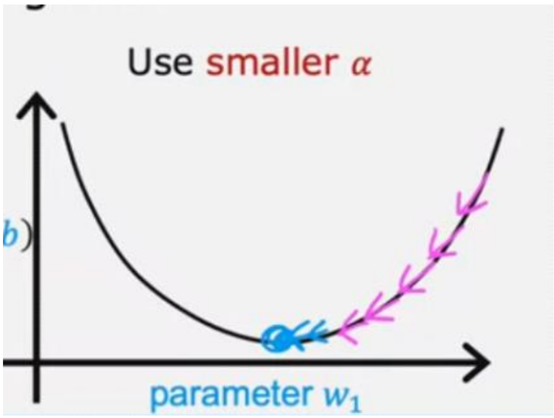
If your learning curve does this:


Learning rate could be to high or bug in code

**If learning rate is too high gradient descent will "bounce" around missing the correct value as it is moving too far each step**




**To fix this, use a smaller learning rate:**



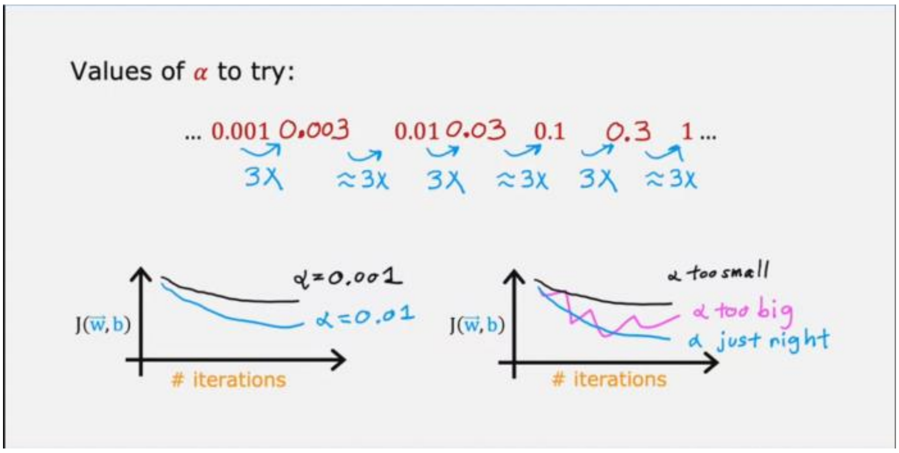

## Tips:

**With a small enough learning rate (a / alpha) the cost function score should decrease each iteration**

**A way to test gradient descent is to set your learning rate really small to see if everything is working or if there a bug**

**Obviously afterwards, increase it again to reach convergence faster**

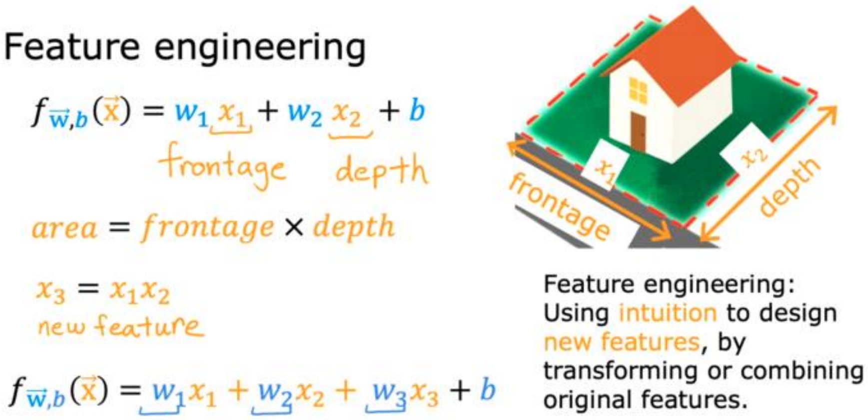
**Pick the learning rate just below the largest possible**


Values of α to try:
... 0.001 0.003 0.01 0.03 0.1 0.3 1 ...
3X ≈3X 3X ≈3X 3X ≈3X
J(w⃗,b)
α=0.001
α=0.01
# iterations
J(w⃗,b)
α too small
α too big
α just right
# iterations

# Feature Engineering

Tuesday, 29 August 2566 BE 08:45

**Choosing the right features (inputs) is a critical step in building a good model**




**Using the existing data to make more useful features, like housing area**

**Transforming and combining existing features to make new ones**

**This increases the accurate of the model**

# Polynomial Regression

Tuesday, 29 August 2566 BE 08:56

Polynomial Regression is a type of feature engineering & Multiple Linear Regression

**Polynomial regression is when you raise one of your inputs to a power or square root**

**BECAUSE OF THIS SQUARING, FEATURE SCALING IS CRITICAL BECAUSE BIG NUMBERS**

**This changes the course of the models prediction**

**Makes none linear patterns**

# Feature Scaling & Learning Rate (Multi-Variable lab)

Wednesday, 30 August 2566 BE 06:58

C1_W2_Lab 03_Featu...

C1_W2_Lab 03_Featu...

Feature Scaling Notes:

Sigma is standard deviation

Getting data around zero is the point

Decreases training time massively

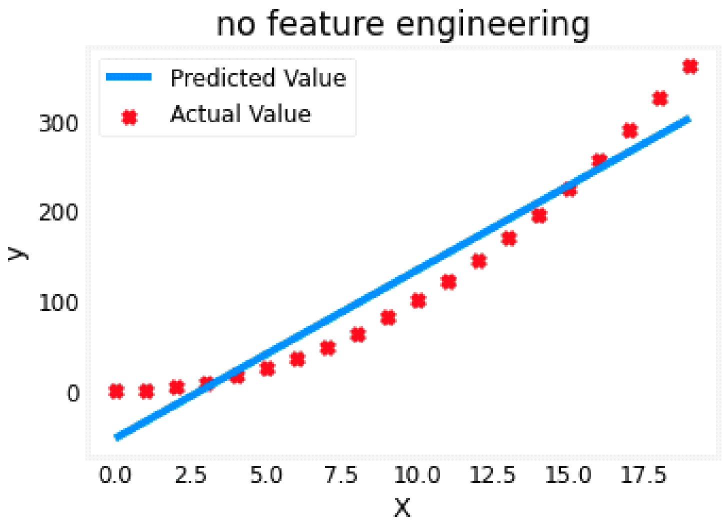
Gets training consistent for all features

Can train with a must higher learning rate

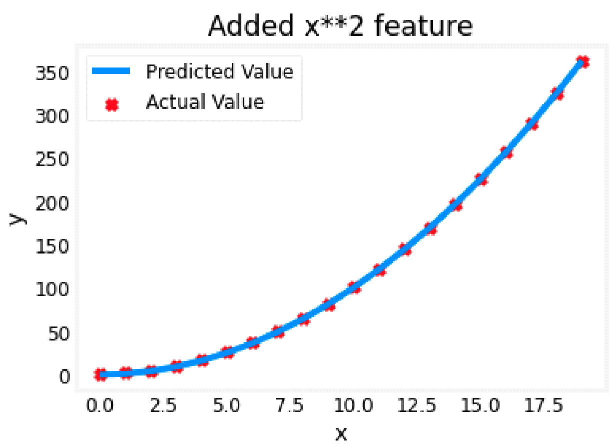
# Feature Engineering and Polynomial Regression (Multi-Variable lab)

Wednesday, 30 August 2566 BE 07:34

Data without Polynomial Regression:




Data with Polynomial Regression:




(Squared the value of each feature)

To find which features may need polynomial Regression use trial and error

**Always pick the polynomial regressions that get you closest to target data**

C1_W2_Lab 04_FeatE...

C1_W2_Lab 04_FeatE...

# Linear Regression using Scikit-Learn

Wednesday, 30 August 2566 BE 08:00

This uses z-score normalization

C1_W2_Lab 05_Sklear...

C1_W2_Lab 05_Sklear...

# Programming Assignment - Linear Regression

Wednesday, 30 August 2566 BE 09:16

C1_W2_Linear_Regr...

C1_W2_Linear_Regr...

Linear Regression lab

Notes from mistakes:

$$= \frac{1}{2m}$$

Means in python:

```
total_cost = cost / (2 * m)
```

Same with:

$$\frac{1}{m}$$

```
dj_dw = gradientWeight / m
```

# Motivations

Wednesday, 30 August 2566 BE 09:45

**Classification is when your models output only has a few choices / classes (y)**

**Classification uses a threshold boundary for making choices (more in later notes.. i assume)**

**YOU CANNOT USE LINEAR REGRESSION FOR CLASSIFICATION PROBLEMS**

## Binary Classification:

**If there is only 2 output (y) choices, its binary classification**

**Often the output of these binary classifiers is "yes / no" or "true/false" or "1 / 0"**

if its "true": it's a positive class

If its "false": it's a negative class

Classification uses a threshold boundary for making choices (more in later notes.. i assume)

**Logistic regression is used for binary classification**

# Classification Basic (Lab)

Thursday, 31 August 2566 BE 06:41

This lab also shows you the linear regression thresholding problem

C1_W3_Lab 01_Classif...

C1_W3_Lab 01_Classif...

# Logistic Regression

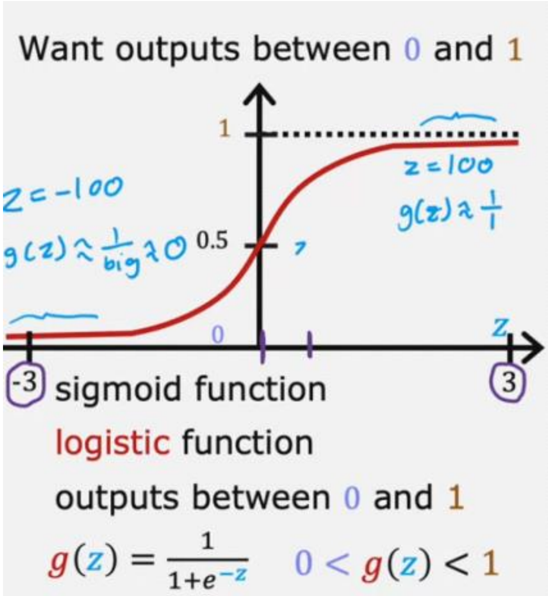
Thursday, 31 August 2566 BE 06:44

**Logistic Regression fits an "S" to the dataset**

**Gives you the probability that a input (y) is = to 1 / true**

**Always outputs a 0 or 1**

Sigmoid Function:




**When z is zero, its sigmoid is 0.5, our decision boundary**

**Outputs values between 0 and 1**

$$g(z) = \frac{1}{1+e^{-z}} \qquad 0 < g(z) < 1$$

E (constant) = 2.7ish

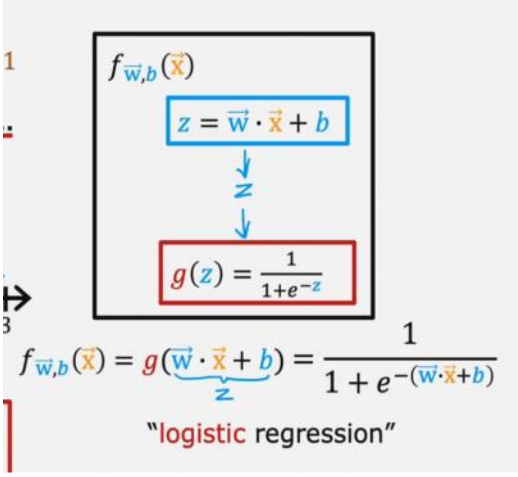
Z is where you are on the "Z" axis

$$0 < g(z) < 1$$

**When "Z" is large, sigmoid its very close to 1, the decision boundary will be near 1**

**When "Z" is very small, sigmoid its very close to 0, the decision boundary will be near zero**

Logistic Regression:




**Z becomes linear regression algorithm (vectored)**

**Outputs a number between zero and one (check left for details):**

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

**Outputs the "chance" / "odds" it's a positive (1) or negative class (0)**

**"e" can be gotten using numpy "np.exp()"**

Research Papers:

$$\underset{P(1)}{} \quad f_{\vec{w},b}(\vec{x}) = P(y = 1 | \vec{x}; \vec{w}, b)$$

**This means what the probability "y = 1" given the input of "x Vector"**

If the value produced by Z is small, likely negative class, if big, likely positive class

# Logistic Regression (Lab)

Thursday, 31 August 2566 BE 07:37

numPy's exp function gives us our e value on command

C1_W3_Lab
02_Sigmo...

C1_W3_Lab
02_Sigmo...

# Decision Boundary

Thursday, 31 August 2566 BE 08:04

Decision Boundary:

$$\text{Decision boundary} \quad z = \vec{w} \cdot \vec{x} + b = 0$$

**To use the model you enter your data into "z"**

**The boundary goes where "z" = 0**

**Decision boundary is when Z is zero always so Z is re-arranged to ensure this**

**So If your "Z" / model =**

**You use "Z" to move the boundary around**

$$f_{\vec{w},b}(\vec{x}) = g(z) = g(\overbrace{\underset{1}{w_1}x_1^2 + \underset{1}{w_2}x_2^2 + \underset{-1}{b}}^{z})$$

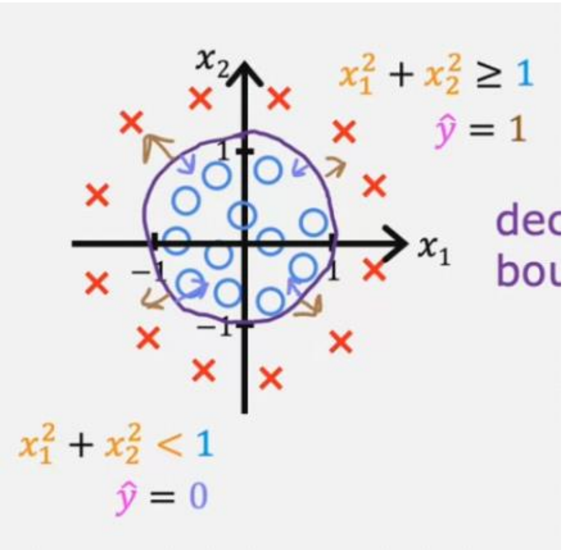
**Decision boundary will be:**

**You re-arrange your model with no inputs to find the decision boundary (z=0)**

**Z is your features, weights / model really**

$$\text{decision boundary} \quad z = x_1^2 + x_2^2 - 1 = 0$$
$$x_1^2 + x_2^2 = 1$$

**Which looks like:**




**Inside the zone Y = 0, Outside Y = 1**

**The boundary will always be straight if no polynomials used**

Logistic Regression:

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

OR

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

Reminder ^

Remember E is a constant (numpy's exp)

If given some data, Z is big, chances are its far outside the bounds

If given some data, z is small, chances are its inside the bonds

# Logistic Regression, Decision Boundary (Lab)

Thursday, 31 August 2566 BE 09:18

C1_W3_Lab 03_Decisi...

C1_W3_Lab 03_Decisi...

"Z" (your model) decides the bounds

It must be re-arranged to always = zero

# Cost Function For Logistic Regression

Thursday, 7 September 2023 06:41

**Squared Cost Function not ideal because lots of local minima for gradient descent to get stuck in (non-convex)**

**// doesn't work for this ^**

Logistic Loss Function:

**A loss function only runs on one example at a time, combine all to make cost function**

## Logistic loss function

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

$f_{\vec{w},b}($

**Fwb is the sigmoid functions output**

Log(F) is for Y = 1

-Log(F) is for Y = 0

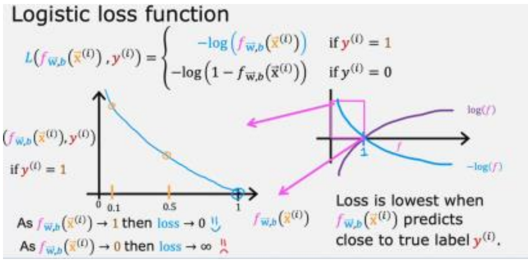
When Y is 1 / positive class:

**Tells you the "cost" for one training example**

**Y = 0 - Negative Class**

**Y = 1 - Positive Class**

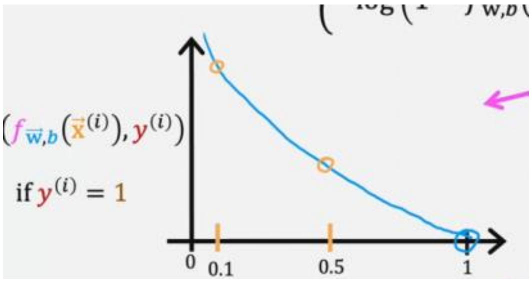
**Logistic Loss functions draws log(f) and -log(f) and uses intersection to measure cost**




**F is the output of logistic regression**

**Log(F) is for Y = 1**

**The further probability is from one (positive class), bigger the loss**




**The bigger the difference between 1 and the real data, the higher the loss**

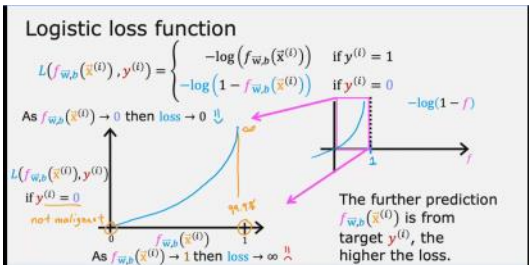
**Loss / cost is lowest when Models probability is closest to true classification**

**COST IS OVERALL DIFFERENCE FOR ENTIRE TRAIINING SET**

**LOSS IS DIFFERENCE FOR ONE EXAMPLE**

When Y is 0 / negative class:

**-Log(1-f) is the graph for negative classes (y=0)**




**The further the probability is from zero, the bigger the cost**

For Log(F) (Y=1):

The closer the probability is from one, lower the loss

For -Log(F) (Y=0):

The closer the probability is from Zero, lower the loss

**This makes a convex function, with a global minimum, so you can use gradient descent**

## Cost

cost

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

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

convex

↳ can reach a global minimum

**REMEMBER, TO COMBINE TOGETHER LIKE ABOVE TO MAKE COST FUNCTION**

# Logistic Regression & Logistic Loss (Lab)

Thursday, 7 September 2023 07:39

C1_W3_Lab 04_Logisti...

C1_W3_Lab 04_Logisti...

**Just like Linear Regression cost, combining the graphs makes the convex graph**

# Simplified Cost Function For Logistic Regression

Thursday, 7 September 2023 07:47

**Fwb is the sigmoid function (logistic regression) output.**

Simplified loss function

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

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

$f_{\vec{w},b}($

**Fwb is the models output - sigmoid function**

**THESE ARE THE SAME ^**

**Using this simplified formula, one side always cancels out**

**Because if Y = 0, this term becomes zero like:**

$$\boxed{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\underset{0}{y^{(i)}}\cancel{\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)} - \underset{(1-0)}{\left(1 - y^{(i)}\right)}\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)}$$

if $y^{(i)} = 1$:

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

if $y^{(i)} = 0$:

$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \quad - (1-0)\log\left(1 - f(\vec{x})\right)$

**If Y = 1, this term becomes zero like:**

$$\boxed{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -\underset{1}{y^{(i)}}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \cancel{\underbrace{\left(1 - y^{(i)}\right)}_{\substack{(1-1)\\0}}\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)}}$$

if $y^{(i)} = 1$:

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

Complete cost function for Logistic Regression:

Simplified cost function

loss

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

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)\right]$$

convex (single global minimum)

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)\right]$$

maximum likelihood (don't worry about it!)

**FINAL BEING:**

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)\right]$$

## MOST USED VERSION (RECCOMENDED):

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

Notes from implementation:

Fw,b is your sigmoid function / what makes your predictions

**You must calculate Z before your able to use it**

**When calculating Z ensure you add Bias to each vector globally after sum product of Weights & Bias**

# Cost Function for Logistic Regression

Thursday, 7 September 2023 08:27

C1_W3_Lab 05_Cost_...

C1_W3_Lab 05_Cost_...

# Gradient Descent Implementation

Thursday, 7 September 2023 08:30

Our Cost function for logistic Regression:

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1-y^{(i)})\log\left(1-f_{\vec{w},b}(\vec{x}^{(i)})\right)\right]$$

Gradient Descent:

repeat {

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b)$$

$$b = b - \alpha\frac{\partial}{\partial b}J(\vec{w},b)$$

**Derivatives**:

**Always calculate the derivative first when implementing gradient descent**

"a" = learning rate

"d thing" = derivative

repeat {

$=1\ldots n$

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b) \qquad \frac{\partial}{\partial w_j}J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}$$

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

} simultaneous updates

$)x_j^{(i)}$

**"xij" is your training data vector**

**Derivatives are not the same as Linear Regression because function definition is different /**

$\left(f_{\vec{w},b}(\vec{x}^{(i)})\right.$

**This bit is different, it's a sigmoid function, not a straight line like linear regression**

**Gradient descent for Logistic regression, you can still use a learning curve like squared linear regression**

Overview:

## Gradient descent for logistic regression

repeat { looks like linear regression!

$$w_j = w_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)x_j^{(i)}\right]$$

$$b = b - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)\right]$$

} simultaneous updates

Same concepts:

- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

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

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

**Still use learning curve to monitor if learning rate is too high or convergence**

**Vectors still work (sum product)**

**Feature scaling (weight too small & standard scales)**

Recommended version:

$$\frac{\partial J(\mathbf{w},b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}\left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}\right)$$

$$\frac{\partial J(\mathbf{w},b)}{\partial w_j} = \frac{1}{m}\sum_{i=0}^{m-1}\left(f_{\mathbf{w},b}(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}\right)x_j^{(i)}$$

# Gradient Descent for Logistic Regression (lab)

Thursday, 7 September 2023 09:16

C1_W3_Lab
06_Gradi...

C1_W3_Lab
06_Gradi...

# Logistic Regression Using Scikit-learn

Thursday, 7 September 2023 09:39

C1_W3_Lab
07_Scikit...

C1_W3_Lab
07_Scikit...

# Completed Logistic Regression

Wednesday, 13 September 2023 19:39

C1_W3_Logistic_Regr... Week 3 practice l... C1_W3_Logistic_Regr...

CHECK DEISGN FLOW FOR MORE DETAILS

**Notes:**

Always add bias afterwards

When calculating Z make sure its done globally for each vector after the sum product of weights & bias

# The problem of overfitting + underfitting

Friday, 8 September 2023 06:32

g

Overfitting - When it fits the training data but makes bad predictions

Underfitting - When it doesn't fit the training data well

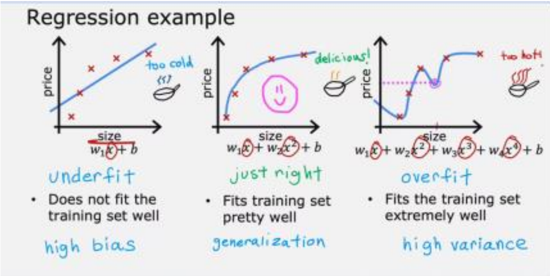
Regression:

**When a model doesn't fit the data well its Underfitting or High Bias**

**When a model fits the training set "too well" its Overfit or High Variance because it can't make good predictions**

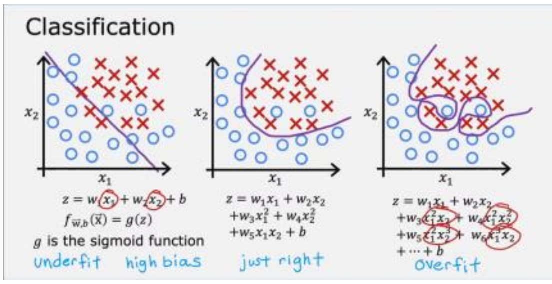
**Normally underfitting is from a lack of features**

**Normally overfitting is because you have too many features**




Classification:

**Overfitting (too much) and Underfitting (not enough) occurs to the decision boundary in classification**



# Addressing Overfitting

Friday, 8 September 2023 06:56

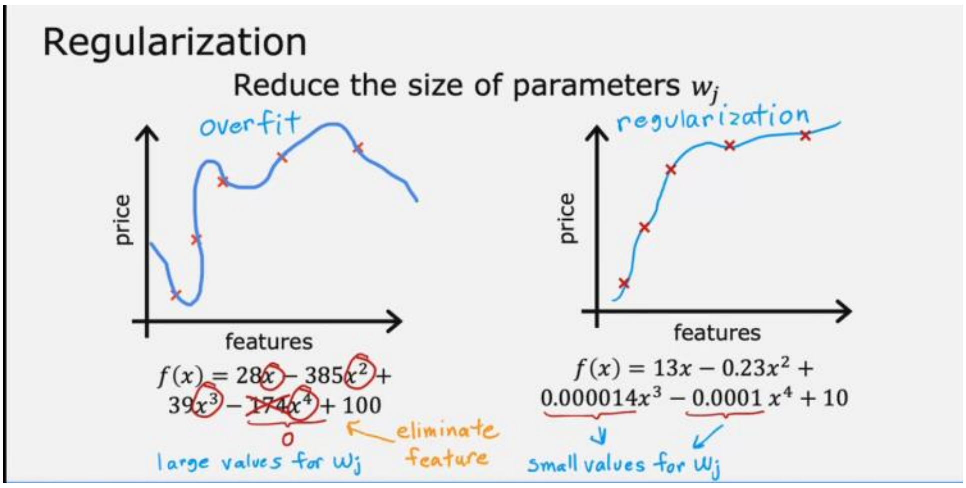
**Overfitting can take place on both your cost function & gradient descent (cost function reccomended)**

**Collecting more training examples will fix overfitting - number one tool**

**If not enough data, cut down on Features**

## Regularization:

**Shrinks the values (using the wight) of some features to reduce there affect - fixes overfitting**



# Overfitting (lab)

Friday, 8 September 2023 07:28

C1_W3_Lab 08_Overfi...

C1_W3_Lab 08_Overfi...

# Cost function with regularization

Friday, 8 September 2023 07:40

Different from Feature Scaling as Feature Scaling occurs to the values of the features, not the weights.

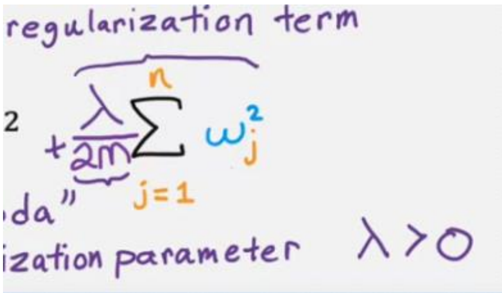
**Overfitting - when the data is too perfectly aligned with training set, so can't predict**

**Underfitting - not aligned enough for training set to make good predictions**

**Apply regularization to all features weight's**

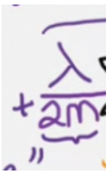
$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \overbrace{\underbrace{\frac{\lambda}{2m}}_{\text{"lambda"}}\sum_{j=1}^{n} w_j^2}^{\text{regularization term}}$$

regularization parameter $\lambda > 0$




**This is a Regularization Parameter, it applies regularization globally to solve overfitting**

**"Lambda" is where you choose your regularization parameter**



**So that both first and second term are scaled, makes it easier to pick good value for "lambda"**

**This also allows you to use same regularization parameter with larger dataset normally**

Lambda Value control:

**If lambda is too big, the model will underfit**

**If lambda is too small, it will still overfit**

**Work on getting it "just right"**

**Increasing lambda, decreases weights**

**Decreasing lambda, increases weights**

# Regularized Linear Regression

Friday, 8 September 2023 08:08

Gradient Descent with Regularization using Lambda:

### Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^{m} \left[ (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

} simultaneous update $j = 1 \ldots n$

$$w_j = \underbrace{1 w_j - \alpha \frac{\lambda}{m} w_j}_{\underbrace{w_j \left(1 - \alpha \frac{\lambda}{m}\right)}_{\text{shrink } w_j}} \underbrace{- \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}}_{\text{usual update}}$$

$$\alpha \frac{\lambda}{m}$$

$$0.01 \frac{1}{50} = 0.0002$$

$$w_j \underbrace{(1 - 0.0002)}_{0.9998}$$

**On every iteration it reduces the cost a bit, that's how this works.**

**Basically, slows the training just enough to prevent overfitting**

**Remember to update simultaneously just like normal**

How we get the derivative term:

### How we get the derivative term (optional)

$$\frac{\partial}{\partial w_j} J(\vec{w}, b) = \frac{d}{d w_j} \left[ \frac{1}{2m} \sum_{i=1}^{m} (\underbrace{f(\vec{x}^{(i)})}_{\vec{w} \cdot \vec{x}^{(i)} + b} - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2 \right]$$

$$= \frac{1}{\cancel{2}m} \sum_{i=1}^{m} \left[ (\vec{w} \cdot \vec{x}^{(i)} + b - y^{(i)}) \cancel{2} x_j^{(i)} \right] + \frac{\lambda}{\cancel{2}m} \cancel{2} w_j \quad \text{No } \sum_{j=1}^{n}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left[ (\underbrace{\vec{w} \cdot \vec{x}^{(i)} + b}_{f(\vec{x})} - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j$$

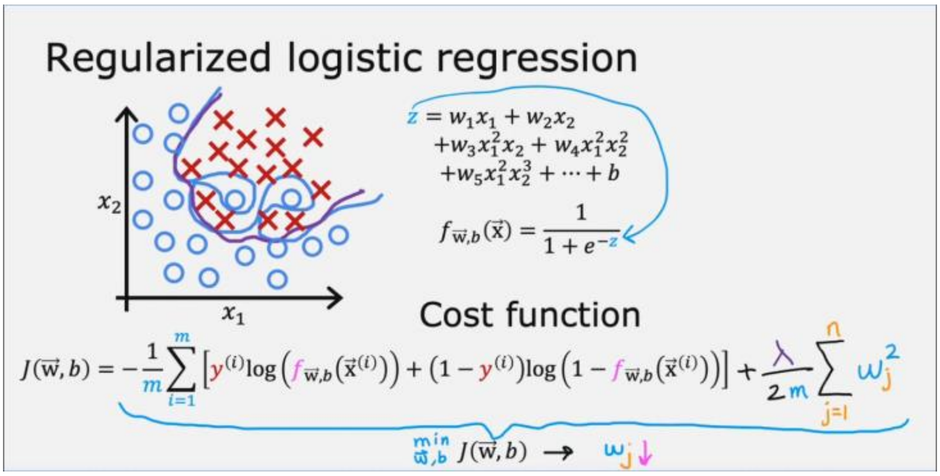
$$= \frac{1}{m} \sum_{i=1}^{m} \left[ (f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j$$

# Regularized Logistic Regression

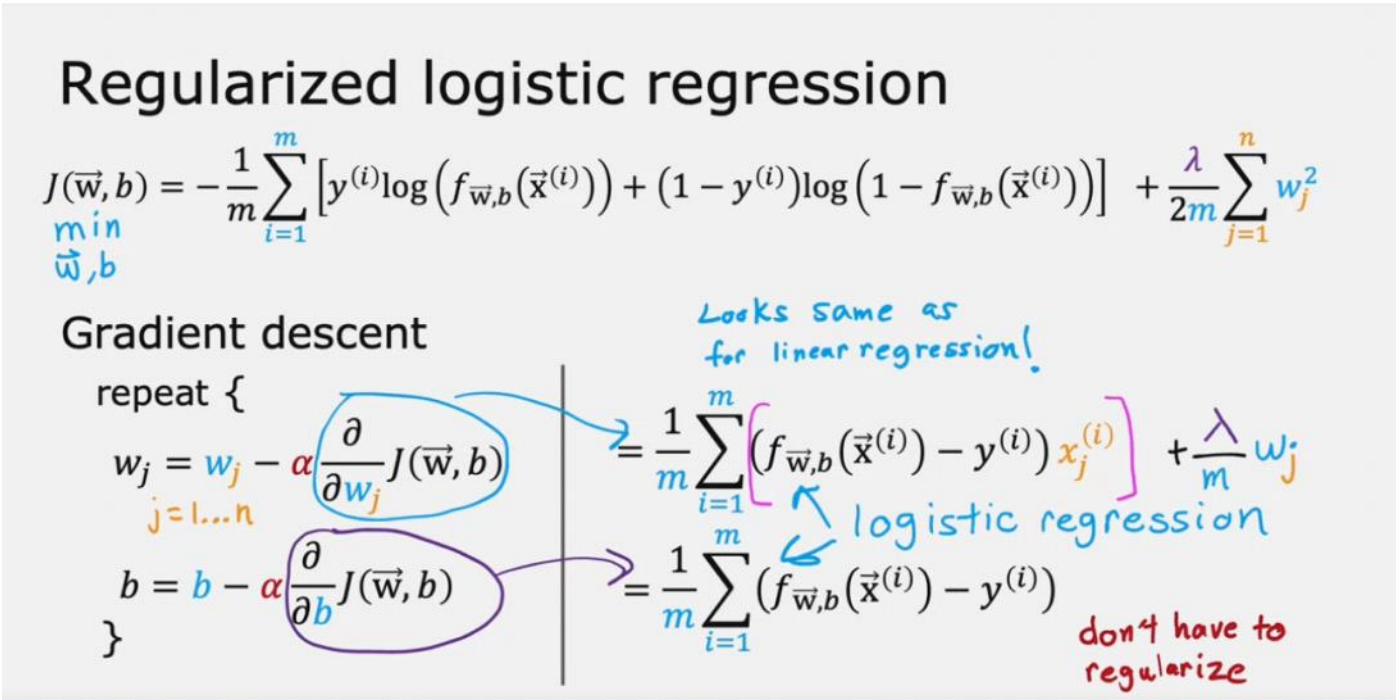
Friday, 8 September 2023 09:08

**Cost function with Reggeization parameter for logistic regression:**

COST FUNCTION:




GRADIENT DESCENT:




**Remember this cost function auto cancels out cuz maths**

# Regularized Cost & Gradient for Logistic & Linear Regression

Friday, 8 September 2023 09:36

You would normally keep going until cost was zero, but this lab isn't complete

C1_W3_Lab 09_Regul...

C1_W3_Lab 09_Regul...

Updated Version -

**Add bias to vectors not per parameter**

**X vector is often training data**

**calculate derivative first for gradient descent**

C1_W3_Log istic_Regr...

# Design Flow

Tuesday, 12 September 2023 19:49

1. Implement model with your learning algorithm function - with input fields as required (**This is fwb)**
   a. Calculate your inputs first (z etc)
   b. If multiple place them into vectors
   c. Implement learning algorithm using learning data

2. Implement a cost function in the following order:
   a. Get control of input vectors and understand them
   b. Calculate models input fields (z etc) - Adding bias to each vector once sum product of weights and bias completed
   c. Implement cost / loss algorithm (remember fwb is your models function)
   d. Average across data & return (normally using (1/m) * total_cost

3. Implement gradient descent

   a. Start by getting prediction (y-hat) / getting fwb
   b. Then calculate gradient / **derivative FIRST**
      i. Add bias gradient to global vector being doing weight gradients

   c. Complete gradient descent by implementing derivative into greater gradient descent code.
   **d. Remember, Weight & Bias must be updated at same time when doing gradient descent**

**4. Make a prediction function for trained model - this means installing thresholds if classification etc (connect everything up here)**

5. Regularization may need to be carried out on the cost function or on gradient descent (recommend cost function) if your model is overfit, check code for details - Regularization is done per vector not per parameter

**Notes:**

Fwb is the output of your model, because F is always your function

Calculate cost function inputs first always (before cost function)

**Add bias after competing weights for each parameter - add bias for overall vector in all cases**

**Bias is always done per Vector NOT per parameter**

# Logistic Regression (with regularization + Vectors)

Tuesday, 19 September 2023 08:44

C1_W3_Logistic_Regr...

C1_W3_Logistic_Regr...

# Simple Completed Logistic Regression - Done awhile ago

Wednesday, 13 September 2023 19:39

C1_W3_Logistic_Regr...    Week 3 practice l...    C1_W3_Logistic_Regr...

CHECK DEISGN FLOW FOR MORE DETAILS

**Notes:**

Always add bias afterwards

When calculating Z make sure its done globally for each vector after the sum product of weights & bias

# Terminology

Wednesday, 16 August 2566 BE 09:01

## Terminology:

Training set: Data used to train a model INCLUDES INPUTS AND OUTPUTS FOR SUPERVISED LEARNING

x = "input" variable OR feature variable

y = "output" variable OR target variable

m = number of training examples

(x,y) = first training example

W (weight) = parameter - what you adjust during training

B (bias) = parameter - what you adjust during training