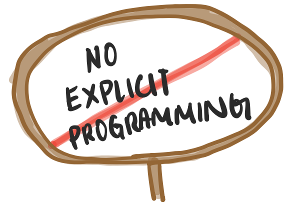
Machine Learning Cartoons

Notes from Andrew Ng Coursera
Stanford Machine Learning course

Illustrated by
Patrick Tran

v2

Q: What is machine learning?

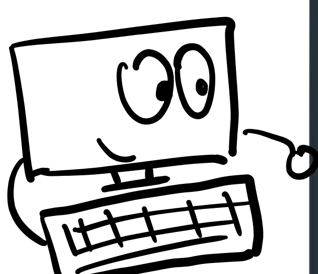


OLDER INFORMAL DEFINITION

The field of study that gives computers the ability to learn without being explicitly programmed

- Arthur Samuel

Example:
Computer **explicitly** programmed to recognize a bear



```
while plugged_in:

  pixel = image[360][368]

  if pixel["color"] === "#8B4513" #brown
    return "brown bear"
  else
    return "a different color bear"
```

Q: What is machine learning?

NEWER MORE MODERN DEFINITION

A computer program is said to learn from experience E with respect to some class of tasks T and performance P

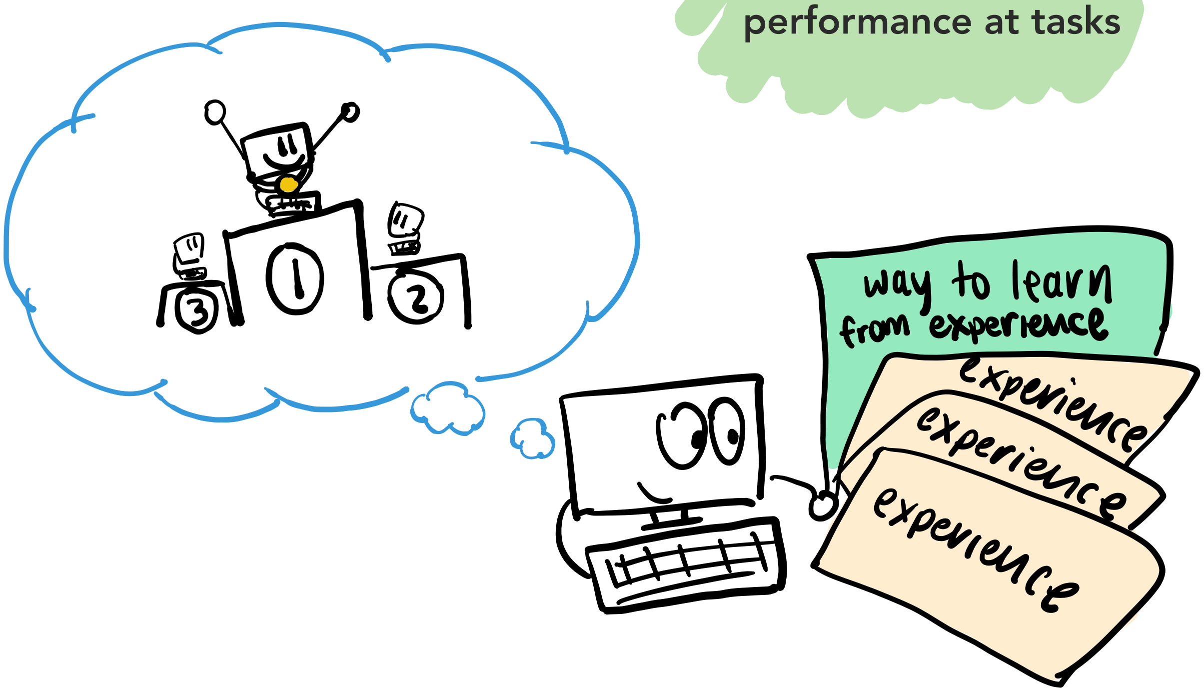
IF

its performance at tasks T as measure by P improves with experience E

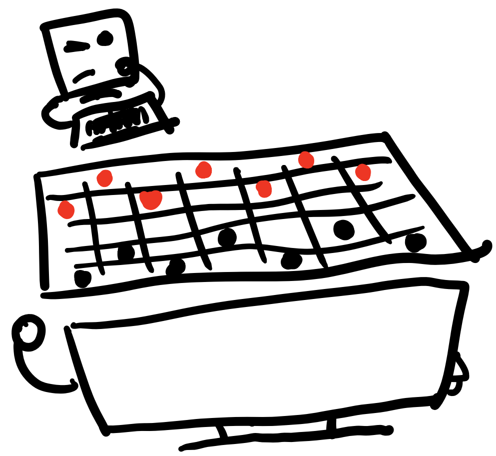
- Tom Mitchell

Example:
Computer program **learning**

In plain English:
More experience leads to higher performance at tasks

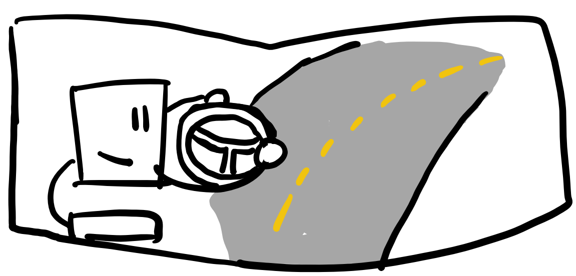


Example:
Computer Program playing checkers



E (experience): playing many games of checkers (or data of games)
T (tasks) : task of playing checkers
p (performance): probability of winning

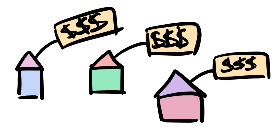
Example:
Computer Program driving vehicles



E (experience): driving many roads (real and simulated)
T (tasks) : task of driving safely from A to B
p (performance): number of accidents

This course covers supervised and unsupervised learning

Supervised learning



We are given data with inputs and correct outputs. For example:

housing data (inputs)
& house prices (outputs)

RAW DATA

x1 Size	x2 Bedrooms	y Price
1000	1	$100k
1750	3	$800k
1500	2	$700k
⋮	⋮	⋮

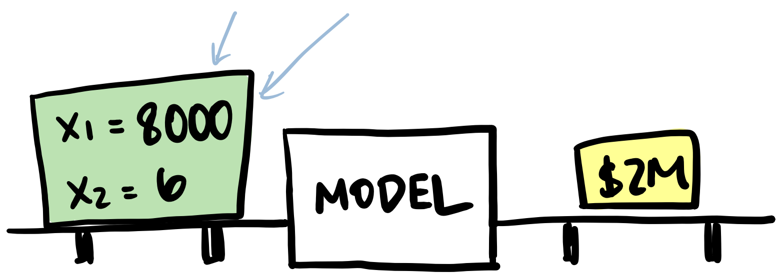
these are inputs (x1, x2) — output (y)

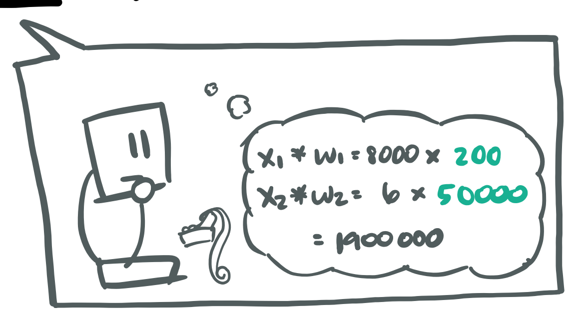
This data is used to create a statistical model (or just "model") used to predict new data

① train(inputs, output) ⇒ model

② model (new housing data) ⇒ predicted house price

Say we wanted to predict the price for size=8000, bedrooms = 6



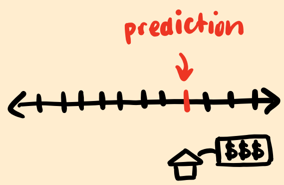


two main flavors of

supervised learning

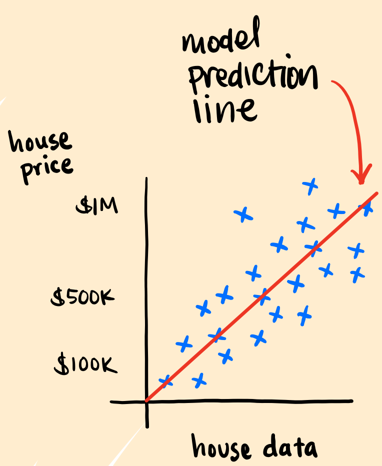
linear regression

predict results in a continuous space



Example 1
given: house data
predicts: house price

Example 2
given: photo of a person
predicts: age of person

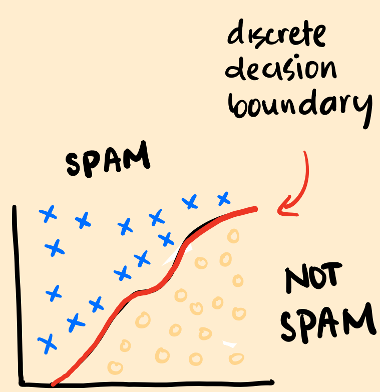


classification

predicts results in a discrete space

Example 1
given: email data
predicts: spam or not spam

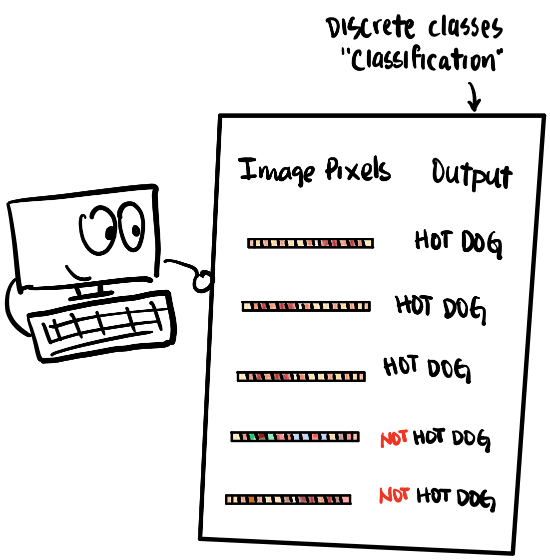
Example 2
given: photo of tumor
predicts: malignant or benign



ART IN LIFE

In an episode of HBO's Silicon Valley, the characters create an app called "Not Hot Dog" that detects whether an image is or isnt a hotdog.

The software engineers behind the show also created this app in real life using a total of 150,000 images to train their model to identify all types of hot dogs



This is a great example of logistic regression!
Inputs are image pixels
Output are the two discrete classes:

1. Hot dog
2. Not hot dog

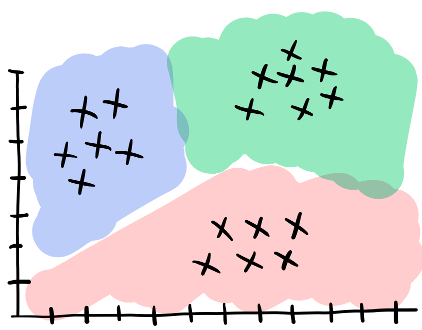
two main flavors of

Unsupervised Learning

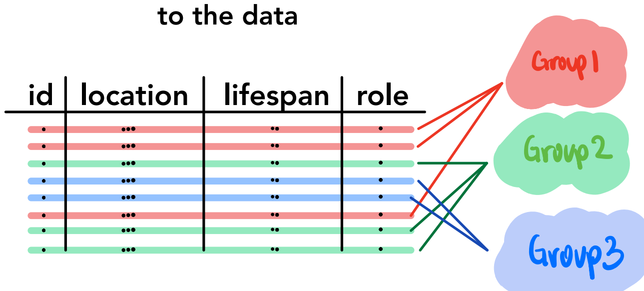
(outputs are not in our provided data)

clustering

"find clusters!"

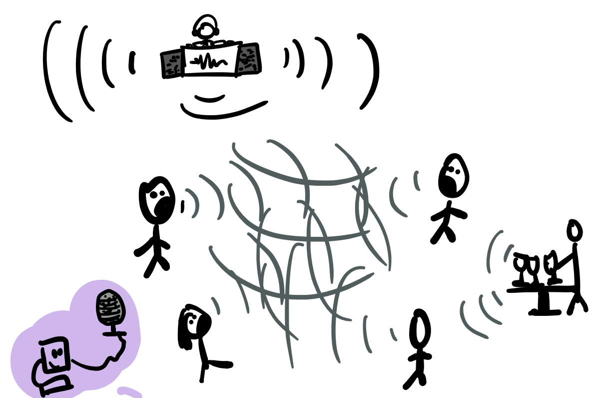


Given data, we use algorithms that find groupings or "clusters" to the data



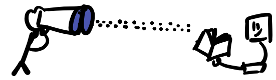
not clustering

example: Cocktail Party Problem

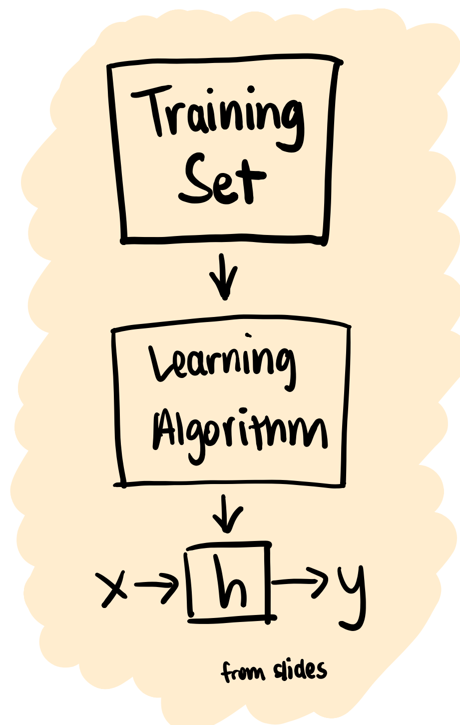


This computer can listen to sounds and identify individual voices and music from a mesh of sounds

Supervised Learning

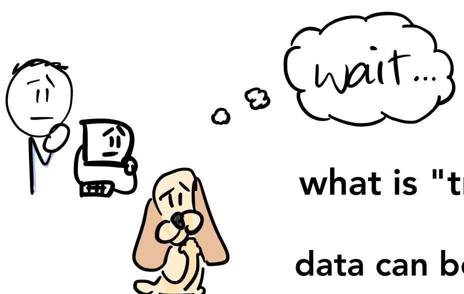


In supervised learning, our goal is to learn a function given a set of data



$$h(x) \rightarrow y$$

Hypothesis function (h) given input (x) predicts output (y)



what is "training set"?

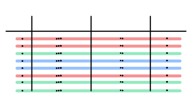
data can be used for training our model...

🤯 data we decide to use for training is "training data"

🤯 this set of data, our "dataset" can be referred to as our "training set"

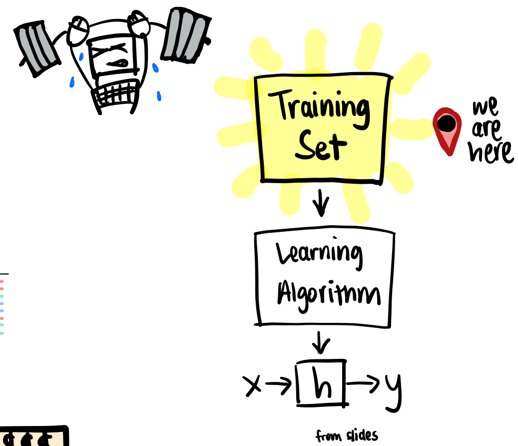
🤯 each individual data attribute of the data can be referred to as a "feature" (number of bedrooms and sq. feet are features of the data)

🤯 so sometimes the training dataset is also the feature dataset, feature data or simply our "features"



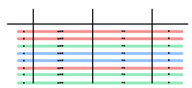
training set
& feature set
& training data
& training dataset
& feature set
& feature data
& features

Training Set



Our data is our training set

Q: What makes up our data?

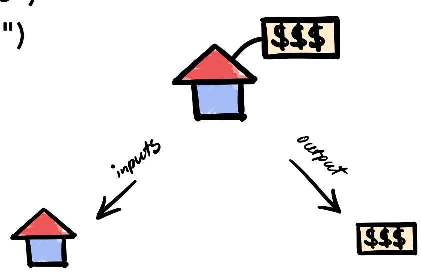


A: In supervised learning, our data has:

1. inputs (our "features")
2. outputs (our" labels")

For our housing example

inputs (features):
x1 = square feet
x2 = bedrooms



output (labels):
y = house price

inputs		output
x1 Size	x2 Bedrooms	y Price
1000	1	$100k
1750	3	$800k
1500	2	$700k
⋮	⋮	⋮

There could be as many variables/ features that your budget and computing power allows!

These outputs of price are in a continuous space so you can tell this data is fit for linear regression.

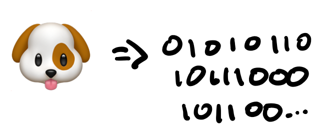
$(x^{(i)}, y^{(i)})$

This pair represents a single training example where the "i" represents the index

m = number of training examples

Classification (logistic regression) would require "classes" as output ie. (can afford / cannot afford)

NOTE: all inputs can be converted to numeric, even images!

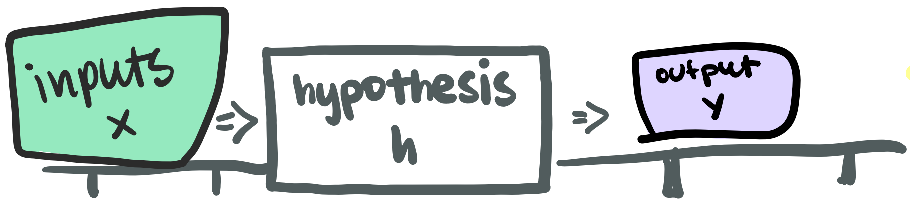


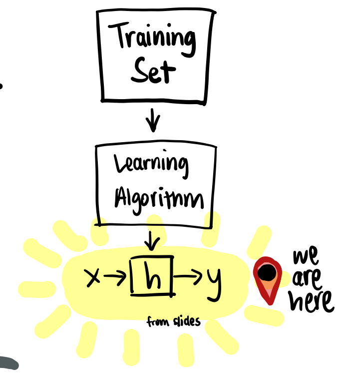
HYPOTHSIS FUNCTION (h)

Q: what does the hypothesis function and look like?

A: All data inputs are eventually converted to numbers and h is a function of these numbers.

$h(x) \rightarrow y$

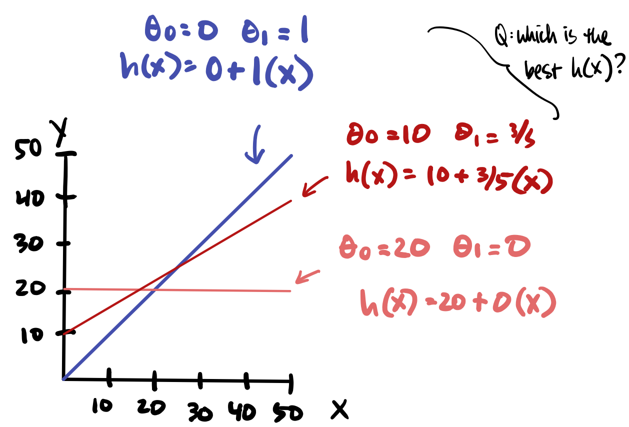




Example h with one input (x) variable

$$h_\theta = \theta_0 + \theta_1 x$$

h functions have a bias value (θ_0)
we want to solve for θ_0 & θ_1
so that h(x) "fits" well with
our training set



remember y=mx+b from high school?

Remember, our data can have one, two, three or even hundreds or thousands of inputs!

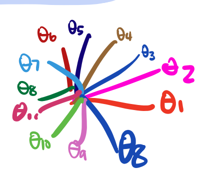
$2 \Rightarrow h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

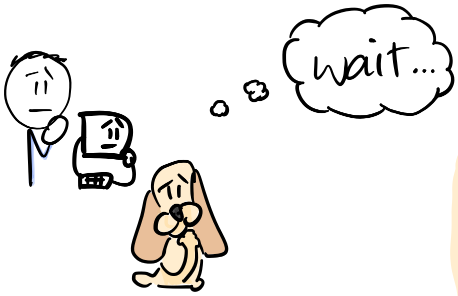
$3 \Rightarrow h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

⋮

$100 \Rightarrow h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5$

this many dimensions becomes hard to graph
so this course usually visualizes 2 or 3 features





"we noticed $h, h_\theta, h_\theta(x)$ being used"

These all refer to the hypothesis function

h_θ clarifies that h is parameterized by theta

$h_\theta(x)$ clarifies that it takes input (x)

Note: hypothesis functions dont need to be simple functions

Depending on our domain knowledge about the data and the problem, we can create complex hypothesis functions

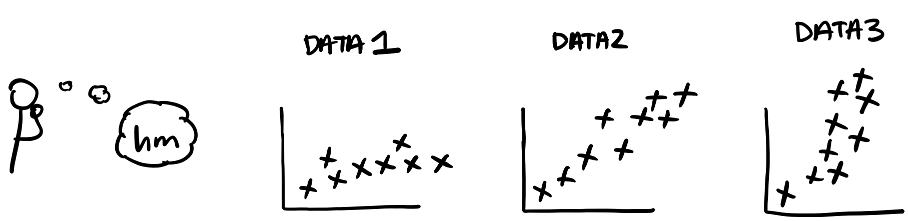
Example: two inputs with a special relationship

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_1 x_2 + \theta_3 x_2$$

Example: two inputs and higher order relationships

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x^2 + \theta_3 x^3$$

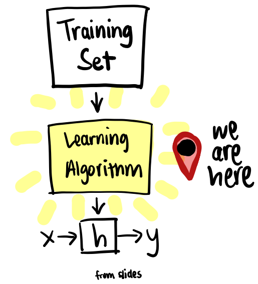
"whichever function we use will depend on our knowledge about the data"

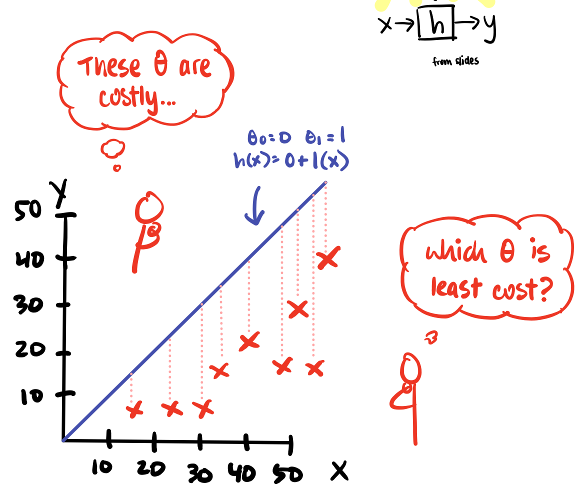


PROTIP: we can try different functions to see which one fits the best

co$t function

Q: How do we evaluate our hypothesis function?
Q: First, <u>why should we</u> evaluate h?
A: We are using h for predictions and we should know how good or bad those predictions are?





A: We evaluate a hypothesis function with a "cost function". Remember, in supervised learning our data includes the correct outputs we can use hypothesis funciton outputs. There are different cost functions but the most common one is "Average Squared Difference"*

Our goal is to find theta (θ) that leads to the lowest cost ($J(\theta)$)

Linear Regression Cost Function

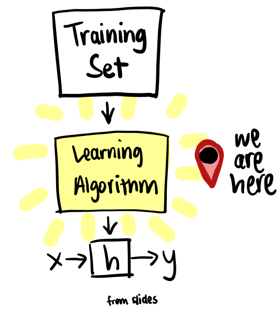
$$J(\theta) = \frac{1}{m}\sum_{i=1}\left(h(x)-y\right)^2$$

averaging factor — sum — difference — square

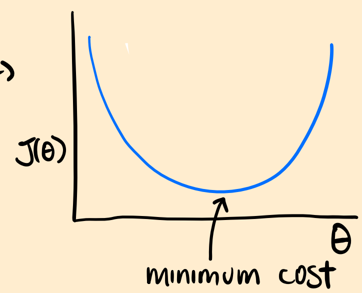
Note: this function amplifies larger errors since an error of 1 is 1^2=1 while a diff of 5 is 5^2=25!

Gradient Descent

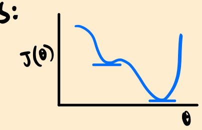
Q: what is gradient descent (GD)?
A: GD is an algorithm that allows us to find the θ (theta) values that lead to the lowest J(θ) (cost).



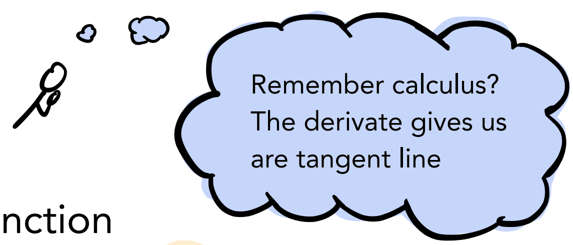
The cost function of linear regression → could look like this



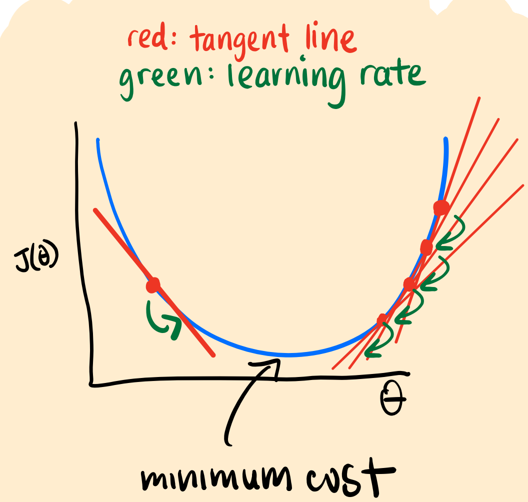
Note: linear regression will always be convex and doesnt have local optima like this:



Let's use the derivative



Q: How?
A: The derivative of the cost function informs us of the slope of the tangent line. with this we know which direction to "descend". The "learning rate" determines the step size.



Gradient Descent Algorithm

(one input example)

This is the condensed version with two thetas: one input theta and one bias theta

learning rate

partial derivative

repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

}

perform simulaneously for all j

Note: technically when there are more parameters we are using the partial derivative

This is the expanded partial derivative from calculus

$$\boxed{\frac{\partial}{\partial \theta} J(\theta) = \frac{1}{m} \sum_{i=1} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}}$$

Note: this a general version of the partial derivative

The final version of the algo has this expanded partial derivative and since $x^{(0)}$ is always 1, we often see it disappear in the formula

Gradient Descent Algorithm - one input

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

}

update simutaneously

week 2


WEEK2 WEEK2
WEEK2 WEEK2
WEEK2 WEEK2
WEEK2 WEEK2
WEEK2
WEEK2
WEEK2

Multiple variables

Our data may contain

(A) single feature or (B) multiple input features

MORE FEATURES MORE POWER!

here: size

X size	Y price
1000	$100K
2000	$200k
3000	$300K
:	:

here: size, #of bedrooms, #floors, age in years

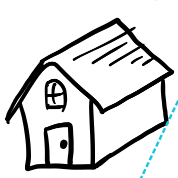
X size	x2 bedroom	X3 floors	x4 age	Y price
1000	1	1	60	$100K
2000	2	1	15	$200k
3000	3	2	48	$300K

$x^{(2)}$

$x_2^{(3)}$

single feature

size?



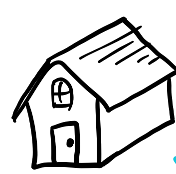
multiple input features

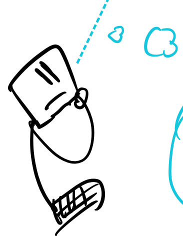
size?

bedrooms?

floors?

age?





you want me to predict price on this single feature???

Notation

n = number of features

$x^{(i)}$ = features of ith training example

$x_j^{(i)}$ = value of feature j in ith training example

Hypothesis Function
Multi-variate + Vectorized

Single feature hypothesis function

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Multiple Variable hypothesis function

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

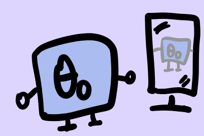
four features

or generally

$$h_\theta(x) = \theta_0 + \theta_1 x_1 \dots \theta_n x_n$$

Note: we have more θ variables than X because of the θ_0 bias value

why is Xo =1?

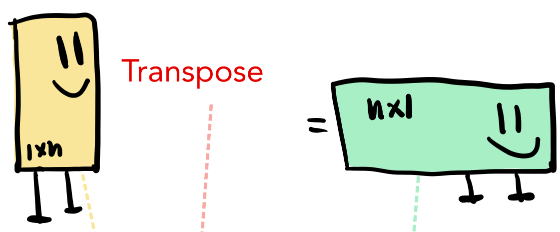


Xo is just a placeholder and 1 multiplied by any value is that same value

In order to create a vectorized representation of hθ(x) we can assume an x_0 value (which is always 1)
Now our vectors both have length n + 1

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \qquad x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

In order to multiply two column vectors we need to transpose a vector



for example

$$\theta^T = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}^T = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \dots & \theta_n \end{bmatrix}$$

Multiplying these two vectors
$h\theta(x) = \theta_0 x_0 + \theta\ xx$
can be represented as

Multiple feature hypothesis function

$$h_\theta(x) = \theta^T x$$

Gradient Descent

Gradient Descent works similar going from one feature to multiple features

In essence, for a single feature, perform a simultaneously update for θ in order to minimize cost. Here, we only looked at θ_0 and θ_1

Gradient Descent Algorithm - one input

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

}

update simutaneously

For multiple features, we are expanding to θ_2, θ_3, all the way up to θn

Gradient Descent Algorithm-multi input

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$\vdots$$

$$\theta_n := \theta_n - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

}

update simutaneously

we can more generally represent 0, … n as j and create this concise function

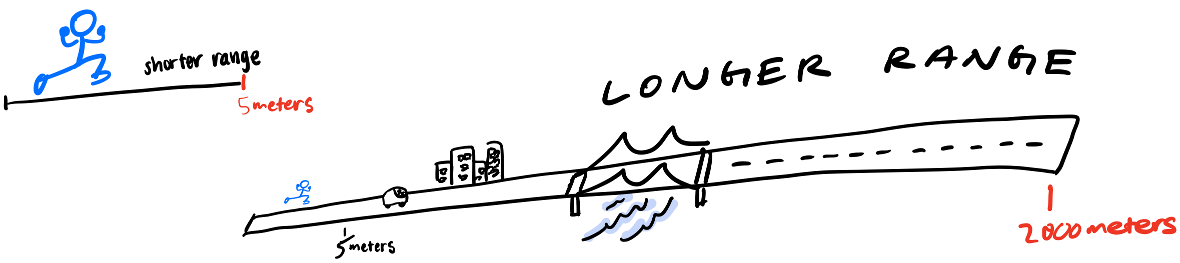
more generally

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Feature Scaling

Gradient Descent in Practice I

What: make sure our features are on a similar scale
Why: our goal is to make gradient descent run much faster
as θ will "descend" quickly on smaller ranges
and slowly on long ranges

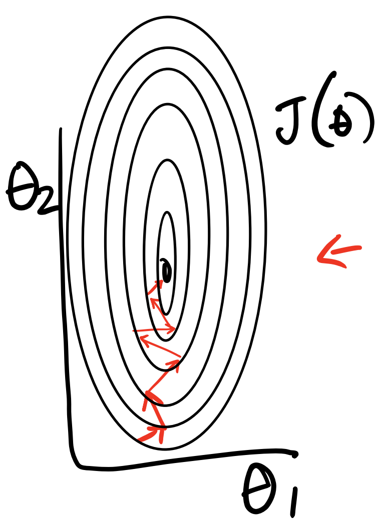


Say you have two features
x_1 = size (0-2000 feet)
x_2 = #of bedrooms (1-5)

Feature scaling:
divides input values
by its range

$$\boxed{\frac{\text{value}}{\text{range}}}$$

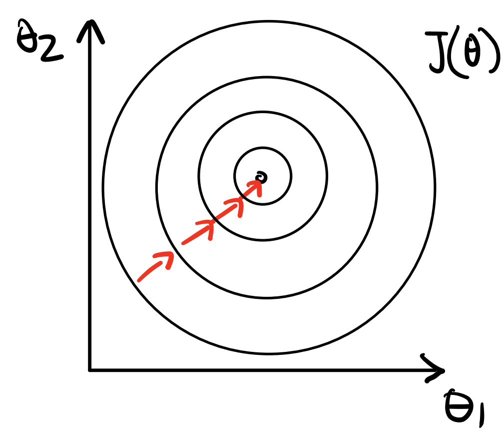
The difference in <u>scale</u> creates
a skewed cost curve graph



gradient
descent
may take
super
long

$$x_1 = \frac{\text{size}}{2000} \qquad x_2 = \frac{\#\text{ of bedrooms}}{2}$$

Which changes our cost curve to:



Two tricks are:

1 feature scaling 2 mean normalization

1 **Feature scaling** involves dividing our feature value by its range in an attempt to shrink its range to

$$-1 \leq x \leq 1$$

Note: every researcher has their own rule of thumb for this range. Ng suggests -3 <= x <=3 and -1/3 <= x 1/3 are also appropriate ranges

2 **Mean normalization** is an additional option that replaces the feature value with feature value minus the mean so the new mean is roughly 0

How to **Mean Normalize**
In the above example if:
Average size x_1 = 1000
Average bedrooms x_2 = 2

$$\text{new (mean normalized) feature value} = \frac{\text{original feature value} - \text{mean}}{\text{range}}$$

$$x' \leftarrow \frac{x - \mu}{s}$$

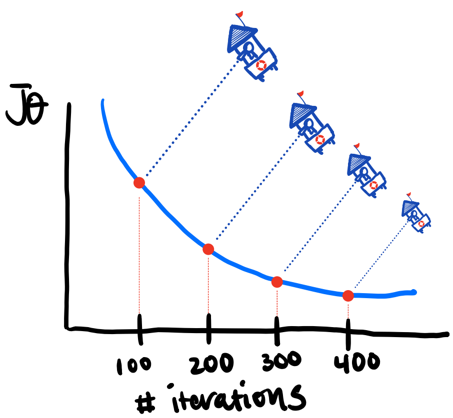
$$x_1 = \frac{\text{size} - 1000}{2000} \qquad x_2 = \frac{\#\text{bedrooms} - 2}{5}$$

Video: Gradient Descent in Practice II

Debugging: How do you know gradient descent is working correctly?

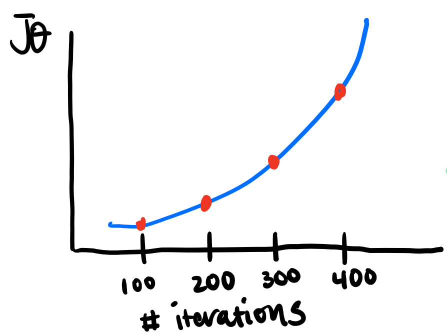
Sometimes GD <u>never</u> converges
Sometimes GD has a <u>slow</u> convergence

Method I: plot of the cost function J(θ) to its number of iterations



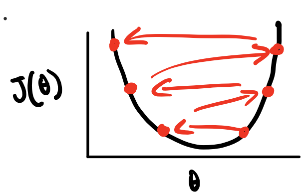
Use checkpoints at 100, 200, 300, 400 iterations to see if gradient descent is working properly.
J(θ) should decrease with every iteration

what if our graph looked like this?

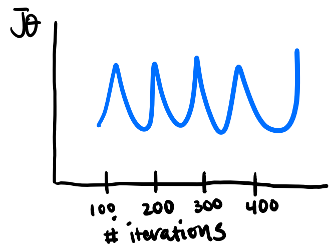


Solution:
Use a smaller learning rate alpha

This means each iteration is making our J(θ) cost larger. This could mean that in our convex curve...



the learning rate alpha is causing our us to overshoot the minimum and actually increasing J(θ) what if our graph looked like this?



Solution: Use a smaller learning rate alpha

Learning Rate (α)

small α	LARGE α
- for sufficiently small α, J(θ) should decrease on every iteration - <u>BUT</u> if α is too small, gradient descent can be slow to converge	- if alpha is too large, J(θ) may not decrease on every iteration AND it may never converge

To choose an alpha try different values:

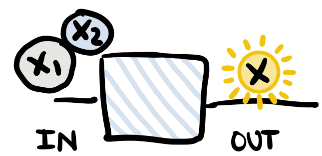
..., 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...

≈ ×3 ×3 ×3 ×3

-ron-a

FEATURES

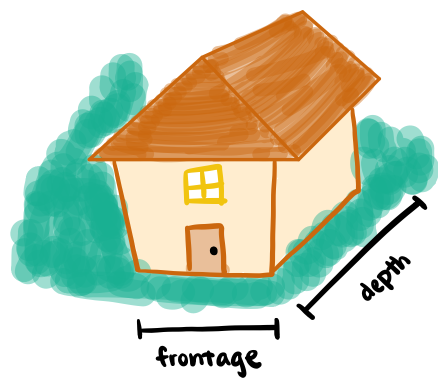
Depending on your domain knowledge about the problem, sometimes defining new features may lead to a better model



Given two features

x_1 = frontage

x_2 = depth



Based on your real estate knowledge you know that "area" is a better predictor of price

area = frontage × depth

Now we can use a better predicting single feature hθ(x)

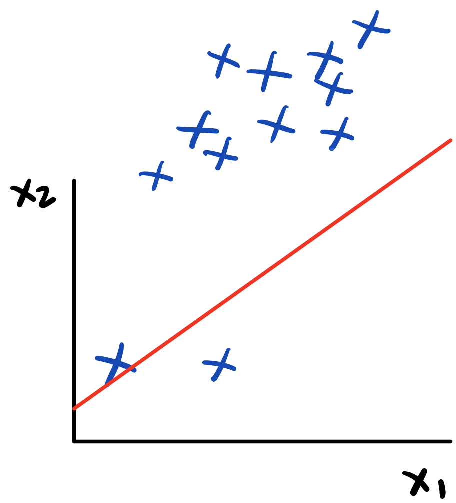
$$h_\theta(x) = \theta_0 + \theta_1 x_1$$

POLYNOMIAL REGRESSION

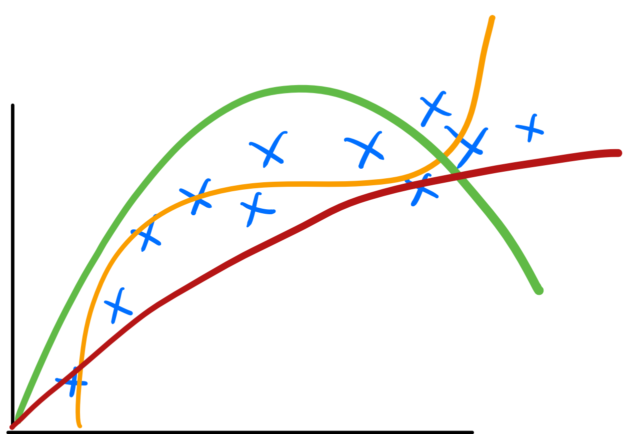
we have been working with straight lines so far

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

no matter what theta is this will represent a straight fitting line through the data



Depending on the data we may want to use a polynomial function



quadratic

$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2}$$

OR

cubic

$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

OR

square root!

$$\boxed{\theta_0 + \theta_1 (size) + \theta_2 \sqrt{size}}$$

Note that feature scaling becomes very important now since ranges get exponentially large!

for scale, if:

$size$ = 1-1,000

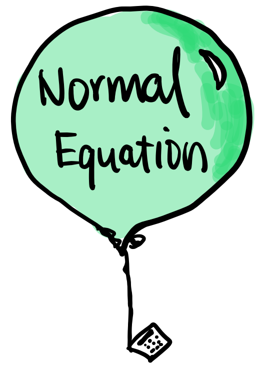
$size^2$ = 1-1,000,000

$size^3$ = 1-1,000,000,000

Parting words:

> "This may be bewildering!
> Which features do I use?
> which equation for the
> hypothesis function?
> Later in the course we
> talk about algorithms
> that will choose features.
> For now just know that you
> can choose different features
> and equations when your
> data calls for it"

Lecture: Normal Equation



we have been using gradient descent so far, alternatively we can use the "normal equation" to solve for theta analytically.

Note: this lecture does not prove why the normal eq works, just how to use it and when to use it

HOW:

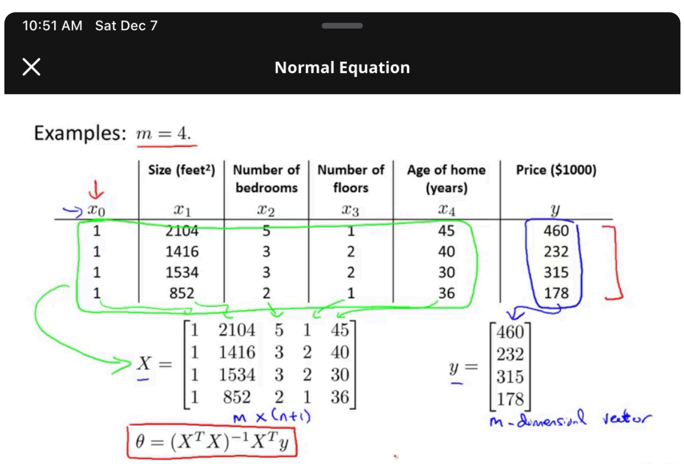
$$\boxed{\text{Normal Equation: } (X^TX)^{-1}X^Ty}$$

where X = inputs

Y = outputs

m = # training examples

n = # features



10:51 AM Sat Dec 7

Normal Equation

Examples: $m = 4$.

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

$$\boxed{\theta = (X^TX)^{-1}X^Ty}$$

Andrew Ng

where $(X^TX)^{-1}$ is the inverse of X^TX

NORMAL EQUATION IN OCTAVE:

```
pinv(X' * X) * X' * y
```

When should we use either?

	Gradient Descent	Normal Equation
Advantage	- Works well even when n is large	- No need to choose learning rate α - No need to iterate
Disadvantage	- Need to choose learning rate α - Needs many iterations	- Need to compute $(X^TX)^{-1}$ - Slow if n is very large

Normal Equation and Non-invertibility

Q: What if $X^T X$ is non-invertible?

A: This should happen very rarely…
But this may be possible because

① Redundant features
Example: you have one feature size (in feet)
and another feature size (in meters)
Solution: delete one of these features

② Too many features
solution: delete features

week 3

Logistic Regression (Classification)

Classification (binary) Examples:

Email: spam, not spam
Online transaction, fraud?: Yes, no
Tumor: malignant, benign
Photo: hot dog, not hot dog

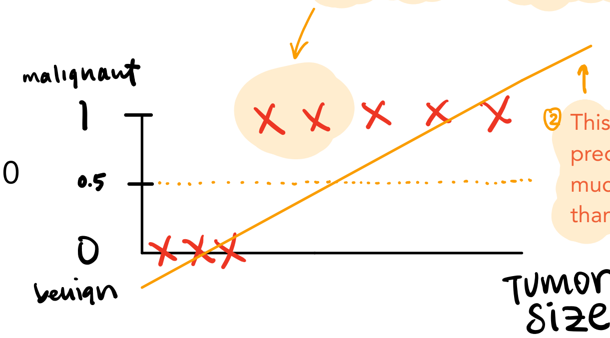
More concisely, O or 1
Given features X, ho (x) = {1, 0}

Note:{1,0} is binary and can be interpreted as {Yes, No}, {SPAM, NOT SPAM}, {X, O} {HOT DOG, NOT HOT DOG}

Looking at a line through data

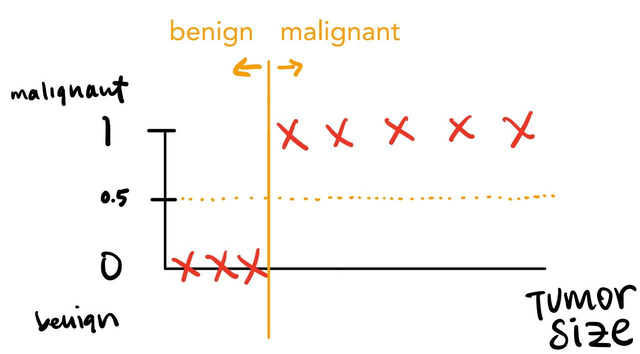
① These are incorrectly identified as 0 (benign) with this best fit line from linear regression

Here we use the best fit line from linear regression:
we may be tempted to interpret
$h\theta(x) > 0.5 = 1$ and $h\theta(x) < 0.5 = 0$



② This line will predict values much larger than 1

we want to go from best fit to best split

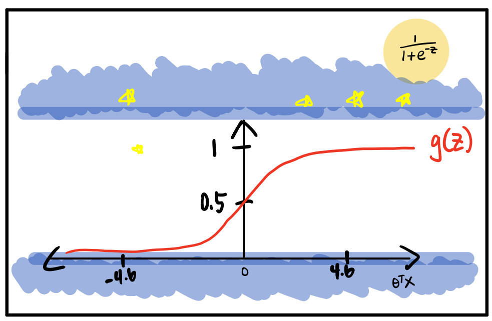


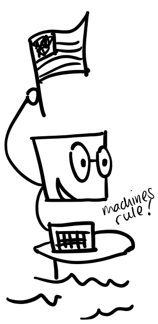
Intuition:

This was our
linear regression
hypothesis function

$$h_\theta(x) = \theta^T x$$

However for **logistic regression**
$h\theta(x)$ should only return {0,1}
so we use a "sigmoid" function (g)
otherwise known as a "logistic" function
to take $\theta^T x$ and fit it to an "S" curve





$\theta^T x$ → sigmoid function → Number between 0 and 1

Logistic Regression Hypothesis Function

$$h_\theta(x) = g(\theta^T x)$$

where

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

one liner:

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

In logistic regression we can think of $h_\theta(x)$ as the probability that y = 1
so if $h_\theta(x) = 0.7$ then there is a probability of 70% that y=1

Notation Check

$$P(y=1 \mid x; \theta)$$

probability

output "y equals 1"

given x

parameterized by theta

More about probability algebra

There is a 100% probability that y = 1 or y = 0
Therefore:

$$P(y=1) + P(y=0) = 1$$

P(DO) + P(DO NOT) = 1

P(SPAM) + P(NOT SPAM) = 1

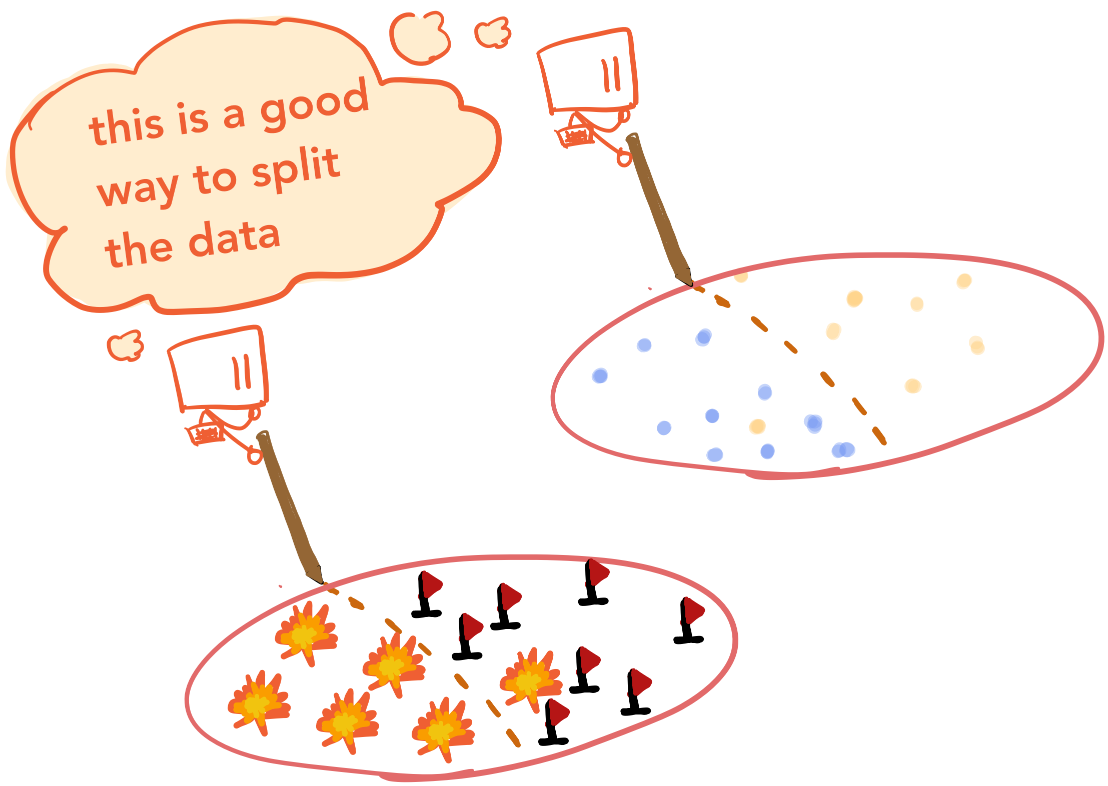
P(🌭) + P(NOT 🌭) = 1

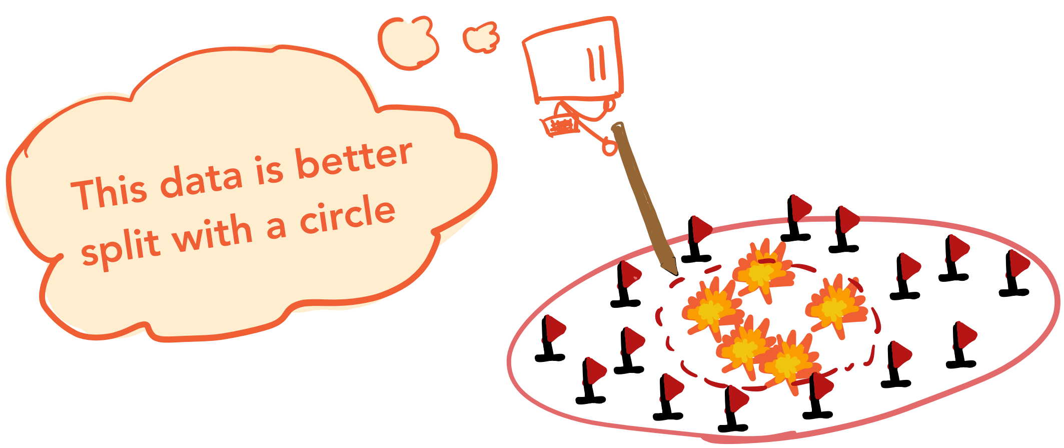
P(NOT 🌭) = 1 − P(🌭)

Note: once you know P(y=1) or P(y=0) you can derive the other through algebra!

Decision Boundary

Finding the boundary to best split the data





Q: How does $h_\theta(x)$ represent a decision boundary?

Say we have...

this hypothesis function and this data

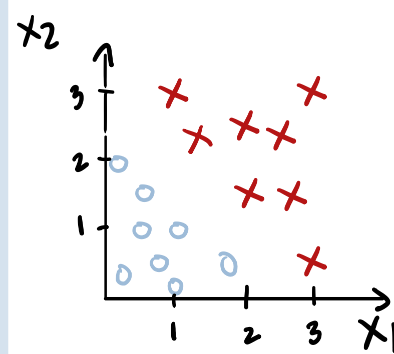
Hypothesis Fn.

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

Data

x_1	x_2	y
1	0	0
0.5	1	0
2.5	2.5	1
1.5	1	0
1	3	1
3	0.5	1
0	2	0
⋮	⋮	⋮

Charted Data



we need to solve for θ(theta) that creates the best decision boundary

✱ we learn how to solve for theta in a later lecture

Imagine we solved for these theta values:

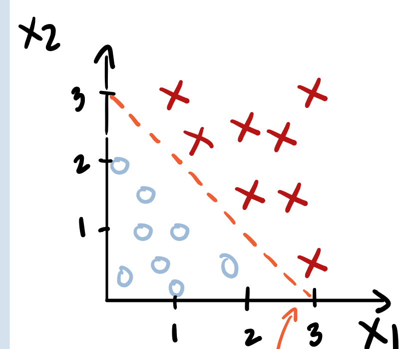
$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix} \quad \text{so} \quad \begin{aligned} \theta_0 &= -3 \\ \theta_1 &= 1 \\ \theta_2 &= 1 \end{aligned}$$

Hypothesis Fn.

$$h_\theta(x) = -3 + 1(x_1) + 1(x_2)$$

and predict $y=1$ if $h_\theta(x) \geq 0$

Charted Data



This is the decision boundary solved by finding theta

$$$$$$$$ LOGISTIC REGRESSION CO$T FUNCTION $$$$$$$$$$$$$$$$$$$$

Q: Can we reuse the **linear** regression cost function as the **logistic** regression cost function?

A: **No**

Q: Why not?

First lets look at our linear regression cost function:

$$\boxed{\text{linear regression } J\theta = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\tfrac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2}}$$

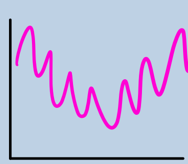
A: Since we have a higher order (non-linear) hypothesis function

$$\left(h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}\right)$$

IF we plugged it into this function our cost graph will be **non-convex**

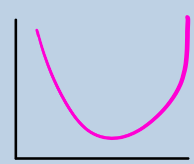
NON CONVEX

This shape means many local optima; gradient descent will struggle to find the best option



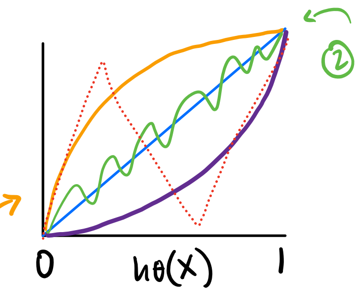
CONVEX

Our goal is to get this pretty looking convex shape that gradient descent can help with



Q: what should the shape of our cost function be?

if y = 0
our cost graph for ha(x)
could be ya of these shapes



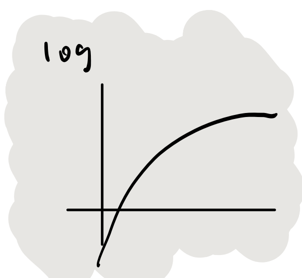
② We know that our cost should go up if y=0 and hθ(x) goes toward 1. Our cost should go up as our prediction is <u>more incorrect</u>. If y=0, hθ(x) = 0.5 should "cost" more than hθ(x) =0.75

① we know if hθ(x) and y=0 we want the cost to be very very low or zero

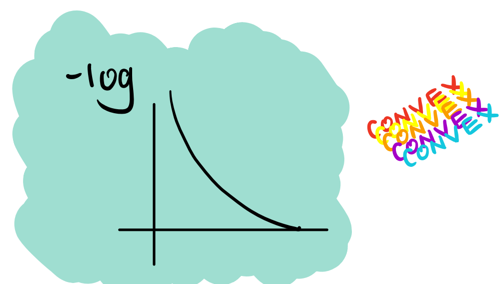
③ We know that this cost curve SHAPE needs to be convex. so gradient descent can work its magic.

A: The log and -log shapes are great shapes

This is the
log function graph



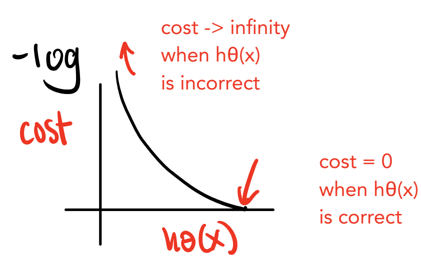
This is the
negative log function graph



Why are these shapes great?

Why is this function great for modeling our cost curve?

- They are convex
- Can have zero cost
- We can model cost to approach infinity (helps to highly penalize wrong predictions)



Logistic Regression Cost Function

TLDR →

$$\mathrm{cost}(h\theta(x), y) = \begin{cases} -\log(\, h\theta(x)\,) & \text{if } y=1 \\ -\log(\, 1 - h\theta(x)\,) & \text{if } y=0 \end{cases}$$

Intuition

if y=0,
we want the correct prediction to "cost" less than incorrect predictions
so…
$h\theta(x) = 1$ to "cost" less than $h\theta(x) = 0$

A TALE OF TWO HYPOTHESIS PREDICTIONS

In notation $\quad$ $h_\theta(x)$ approaches zero

cost $= 0$ if $y=1$ & $h_\theta(x) = 1$

but as $h_\theta(x) \rightarrow 0$ $\quad$ ($h_\theta(x)$ approaches zero)

cost $\rightarrow \infty$ $\quad$ (cost approaches infinity)

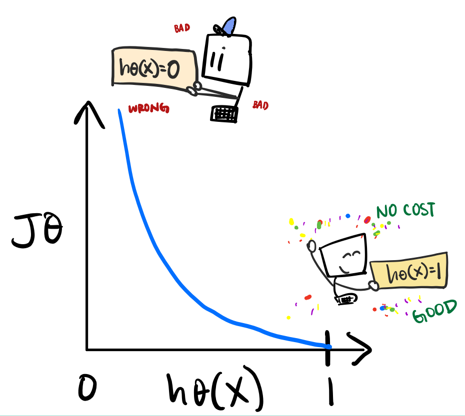
Visualizing the cost graph

Logistic Regression Cost Function

$$\text{cost}(h\theta(x), y) = \begin{cases} -\log(h\theta(x)) & \text{if } y=1 \\ -\log(1 - h\theta(x)) & \text{if } y=0 \end{cases}$$

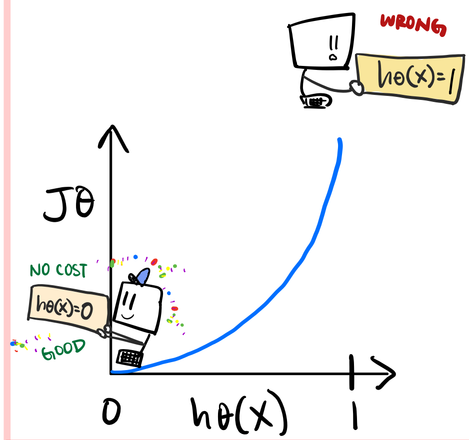
if $y=1$

cost $= -\log(h\theta(x))$



if $y=0$

cost $= -\log(1 - h\theta(x))$



☆ Note that $1-h\theta(x)$ flips the graph because $h\theta(x)$ will be between 0 and 1

Converting our cost function to one line for J(θ)

logistic Regression cost Function

$-\log(h_\theta(x))$ if $y=1$
$-\log(1-h_\theta(x))$ if $y=0$

$$\boxed{\text{cost}(h\theta(x), y) = \begin{cases} -\log(h\theta(x)) & \text{if } y=1 \\ -\log(1 - h\theta(x)) & \text{if } y=0 \end{cases}}$$

$$\text{cost}(h\theta(x), y) = \overbrace{(y) * -\log(h\theta(x))}^{\text{if } y=0 \text{ this portion } = 0} + \overbrace{(1-y) * (-\log(1 - h\theta(x))}^{\text{if } y=1 \text{ this portion } = 0}$$

rearrange (-) sign

rearrange (-) sign

$$-(y * \log(h\theta(x))) + -((1-y) * (\log(1 - h\theta(x)))$$

=

remember transitive property
$-1(x+y) = -x + (-y)$

extract (-) sign

one line!

$$= -1(y * \log(h\theta(x)) + (1-y) * (\log(1 - h\theta(x)))$$

We use this in Jθ to measure average cost

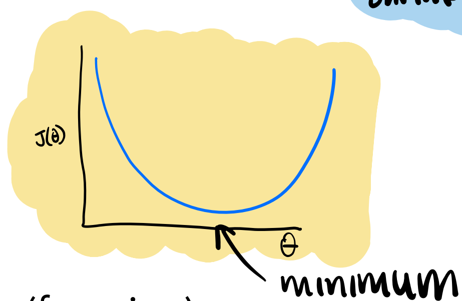
$$J\theta = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(h\theta(x^{(i)}), y^{(i)})$$

$$J\theta = \frac{1}{m}\sum_{i=1}^{m} \left[y^{(i)} \log h\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h\theta(x^{(i)})) \right]$$

Minimizing Cost J(θ)

Like always, given parameters theta (O)
we want to minimize cost

$$\min_{\theta} J(\theta)$$



✯ we have a convex shape (function) and we want to find the min value

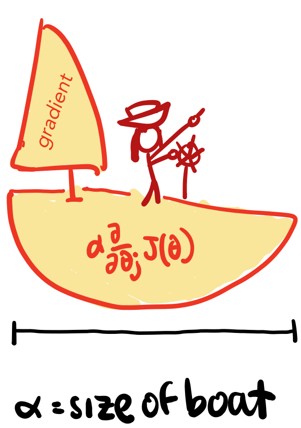
Welcome Back Gradient Descent

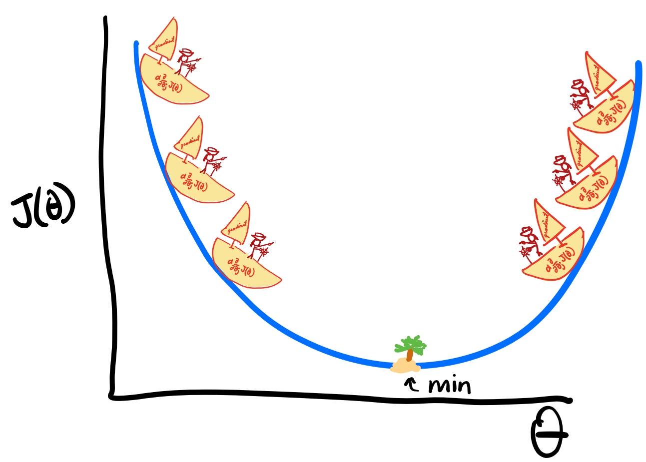
Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$ ← until convergence

}

If the gradient step were a boat...





the gradient boat leads us to the minimum cost!

Expanding $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J\theta$ to something usable

Think back to calculus class

$$\theta_j := \theta_j \, \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Note : θ_j is parameterized by "j" and we have the partial derivative $\frac{\partial}{\partial \theta_j}$

The chain rule from calculus says $\theta_j := \theta_j \, \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ can be rewritten as

$$\theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

Gradient Descent formula

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)}$$

(simulatenously update all θ_j)

If gradient descent were a group of gradient step boat...

Lecture: Advanced optimization

✩ gradient descent alternatives

1. conjugate descent
2. BFGS
3. L-BFGS

* the details of these three are outside the scope of this course

Advantages

- no need to manually pick a learning rate (α) (these algo's will choose α for you)
- often faster than gradient descent

Disadvantages

- more complex

Recommendation:
Do not write these algos yourself
you can use these algo's without fully understanding the implementation

Multiclass Classification

Example classes

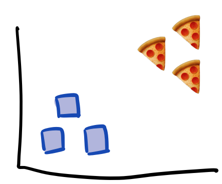
$\underbrace{\text{work, friends, family}}$ multiple classes

email tagging: work, friends, family
y=1 y=2 y=3

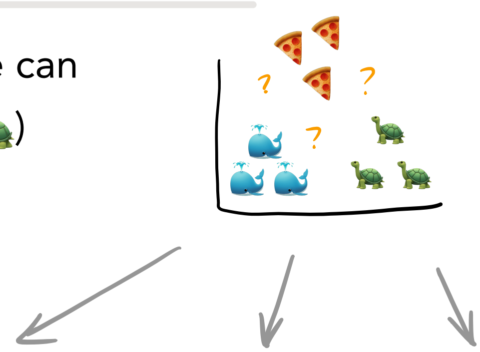
* indices can start at y=0 or y=1

weather: sunny, cloudy, snowy

Instead of binary classification where we only have two classes (true or false, 🍕 or 🐳) …

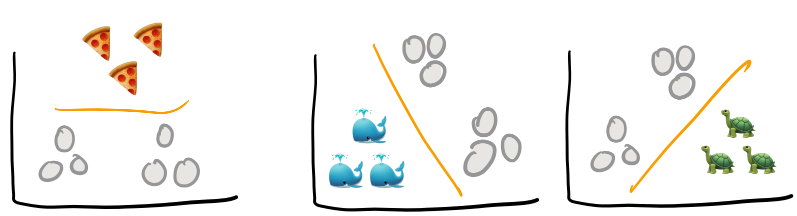


In multi-class classification we can have many classes (🍕, 🐳, 🐢)



We can solve this by creating multiple binary classification problems.

1. 🍕 and NOT 🍕
2. 🐳 and NOT 🐳
3. 🐢 and NOT 🐢



We now have 3 classifiers and for predictions, we run all 3 classifiers and pick the highest score

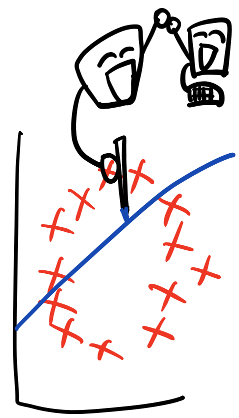
Overfitting & Underfitting

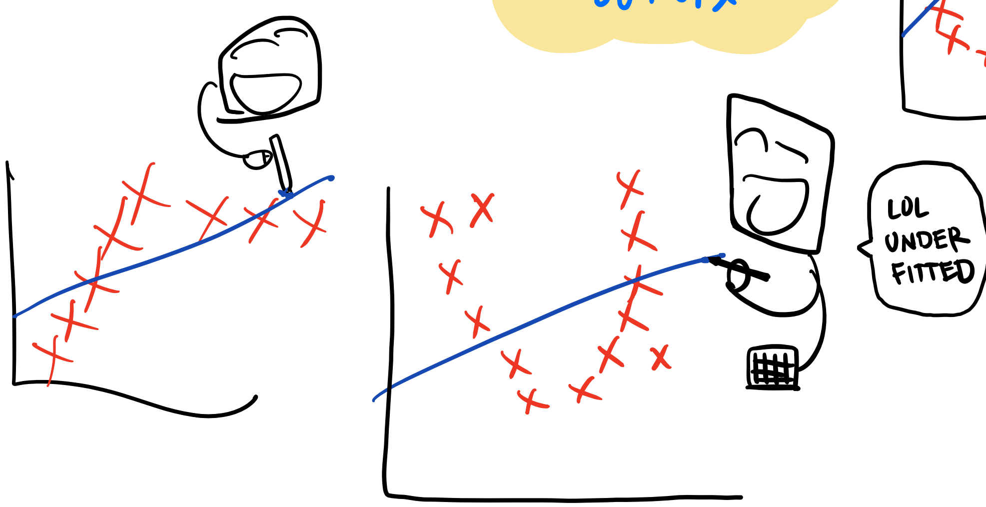
Your model may be improperly fit to the data

① **Underfitting:**
Your model doesn't fit the training data well. "High bias" because similar to your model having strong preconception of the data

Imagine if our models assume all data fit a straight line

$\theta_0+\theta_1 x$



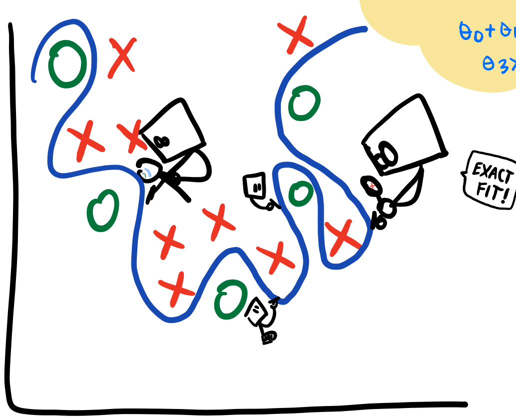


② **Overfitting**
Your model fits the training data <u>too well</u>. It tries too hard to fit the data and fails to generalize to new data

"high variance" ↑

Imagine if the model essentially memorized all the data points of the training set

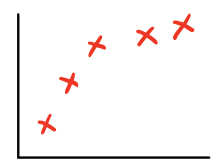
$\theta_0+\theta_1 x+\theta_2 x^2+\theta_3 x^3+\theta_4 x^4 \ldots$

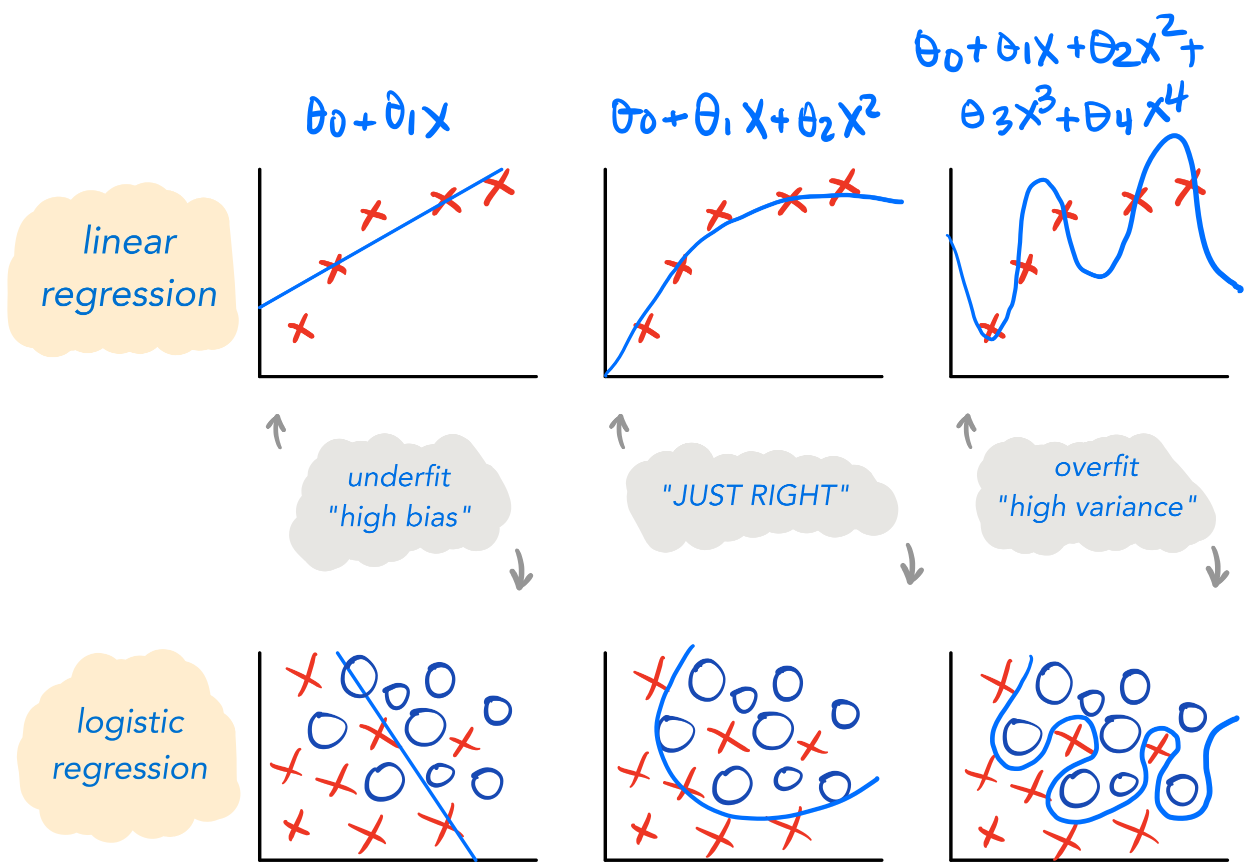


OVER fitting

Q: What is over fitting?

A: If we have too many features, the learned hypothesis may fit the training data <u>so well</u> that it fails to "generalize" to new example inputs

> Note: Given this data we can fit a linear, quadratic, even a *higher* order function →
>
> 



Addressing Overfitting

Its easy to plot data with only a couple features but we will encounter data with many features

x_1	size of house
x_2	color of house
x_3	# of bedrooms
$\vdots$	
x_{100}	cardinal direction of 2nd bedrooms 3rd window

Options

1. Reduce number of features
   - a. manually remove features
   - b. choose a feature selection algo
2. Regularization

Q: **What is regularization?**
The idea behind regularization is that having smaller values for our θ parameters creates a "simpler" hypothesis, smoother functions and is less prone to overfitting

Suppose we want to penalize and make θ values really small. We do this in our cost function by adding an additional term that magnifies the affect to θ

J(θ) = NORMAL COST FUNCTION + REGULARIZATION TERM

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \underbrace{\lambda \sum_{j=1}^{n} \theta_j^2}_{\text{this regularization term will reduce the } \theta \text{ values}}\right]$$

This lambda term (λ) is a parameter we can adjust

LINEAR REGRESSION
W/REGULARIZATION:

LINEAR REGRESSION + REGULARIZATION

LOGISTIC REGRESSION
W/REGULARIZATION:

LOGISTIC REGRESSION + REGULARIZATION

week 4

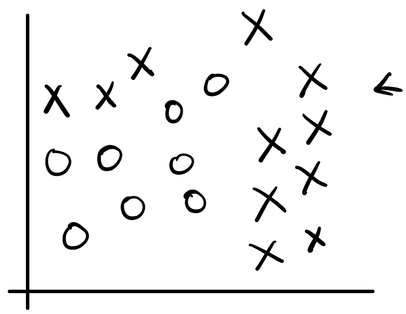
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV
IV	IV	IV	IV	IV	IV	IV	IV

Neural Networks

Intuition

This week we learn about neural networks.
We already have linear regression and logistic regression,
so why do we need another learning algorithm?

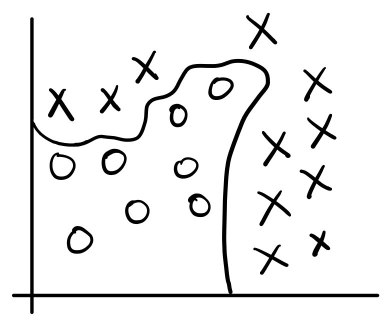
There are situations we want to learn **complex nonlinear hypothesis**.
Consider you have this data.



we could add non-linear terms

$$x_1 x_2 + x_1^2 x_2 + x_1 x_2^2 + \dots$$

When you have only two features we can afford to add all these terms… **but** often you have many more than two features and this becomes computationally expensive **so** we need a better learning algorithm



How do we solve this?

If we used logistic regression, we would add non-linear terms that are complex enough to fit interesting datasets.
Quadratic such as x1*x2 x1*x3 etc…
Or cubic such as x1*x2*x3, x1*x2^2

However, that is a lot of features which leads to

- overfitting,
- computationally expensive

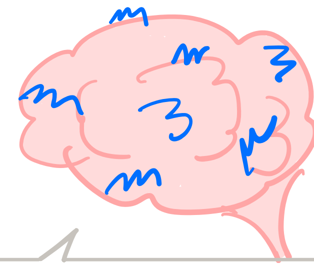
Complex non-linear hypothesis are hard to learn when n is large

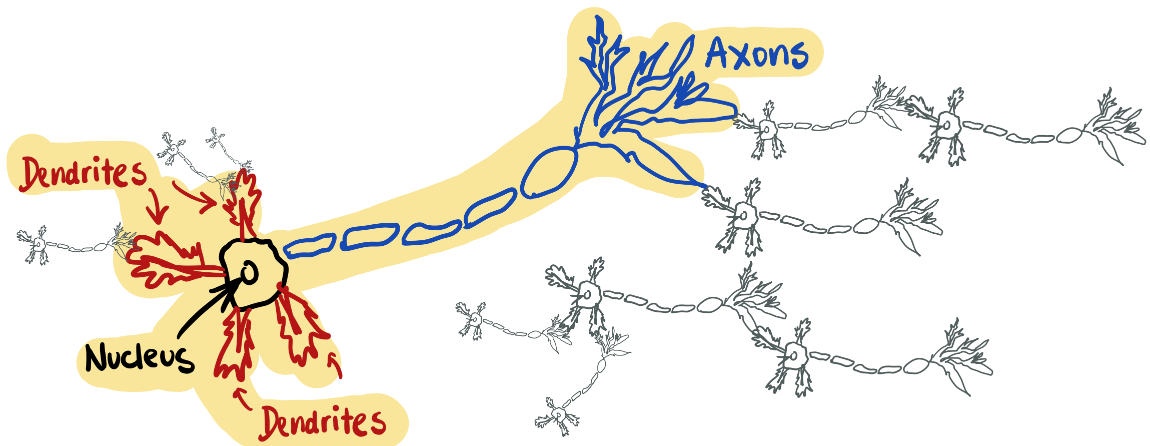
Neurons in the brain

The neurons that make up our brain have

Dendrites - receivers of input

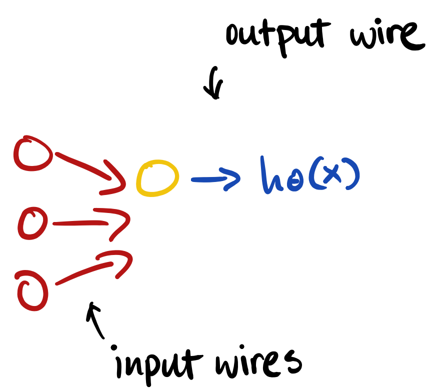
Axons - broadcasters of output





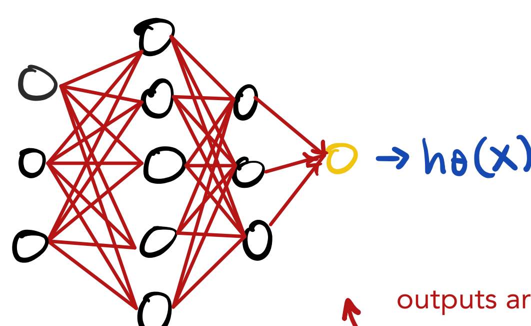
a simple model based on a **single neuron**

can be simplified to:

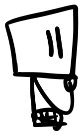


---

a **neural network** is made up of connected neurons

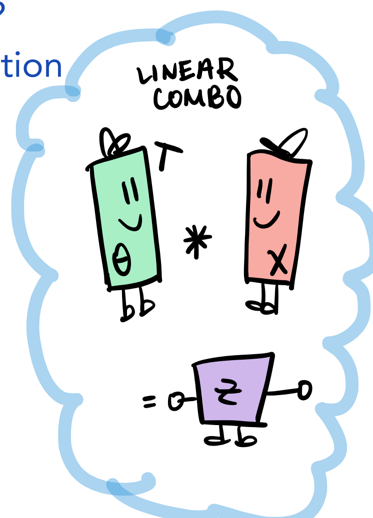


outputs are
inputs to other modes



WAIT...

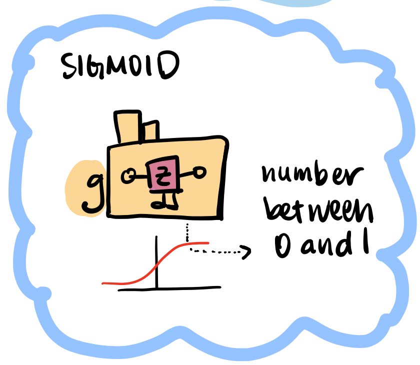
Q: Where does the nonlinearity come from?
A: At each node, $\theta^T X$ is the linear combination of theta (weights) and x (inputs).



We call $\theta^T X$, "z"

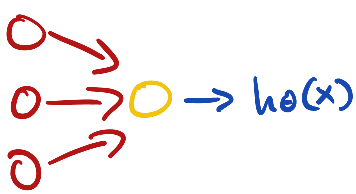
$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \quad X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z = \theta^T X = [\theta_0\ \theta_1\ \theta_2\ \theta_3] * \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

The difference is that
we put z through an "activation function"
which is a nonlinear function "g"



In this course, g will be the sigmoid function.

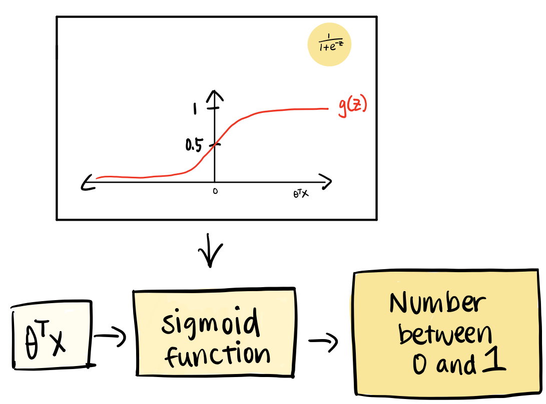
$h_\theta(x) = g(z)$ sigmoid function! $\frac{1}{1+e^{-z}}$



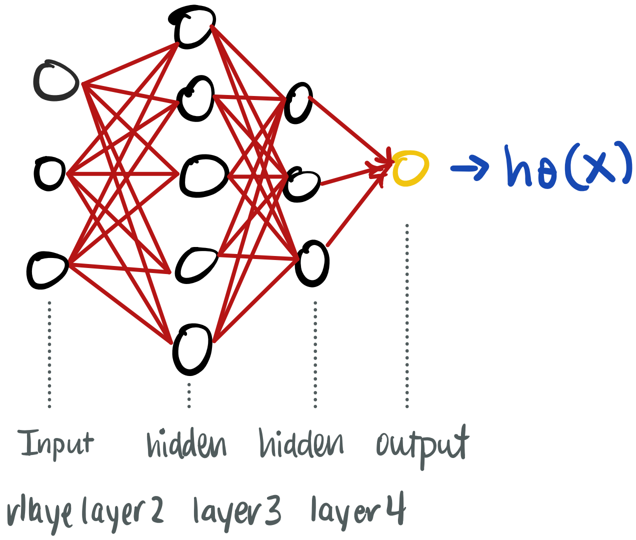
so $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Here "g" is the same sigmoid function from our logistic regression lecture

*In fact, without the network, this should look similar to logistic regression



A neural network is a collection of these neuron calculations.

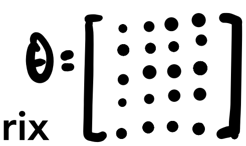


Terminology

The **first** layer is called the **input** layer

The **last** layer is the **output** layer

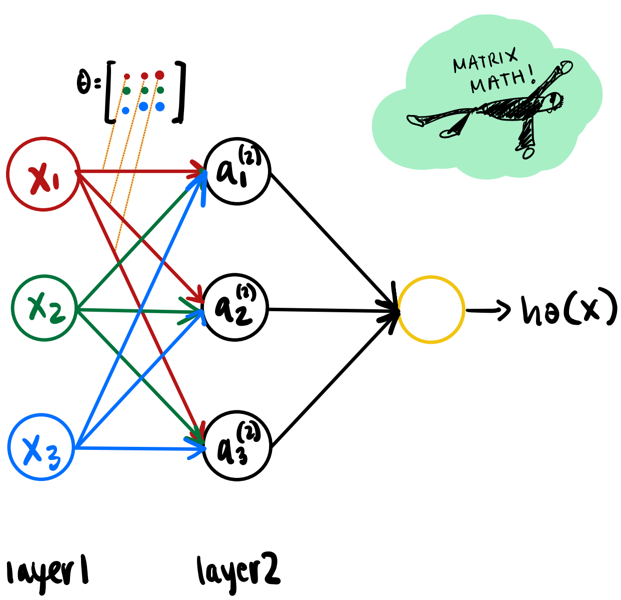
All layers in between are **"hidden"** layers



Note: Theta is now stored in a matrix

In the single neuron example (like in logistic regression), output to a single node means theta is a 1D vector.

In the network,, each layer can output to 1+ nodes for the next layer so theta is a matrix of paramters (or “weights”)



Notation

$a_i^{(j)}$ activation unit of unit i in layer j

$\theta^{(j)}$ matrix of weights going from layer j to layer j + 1

X_n inputs from the data

Notation

This is a single theta value

$$\theta^{\text{from layer}}_{\text{to node}\ \text{from node}}$$

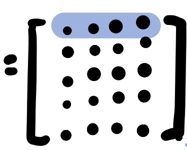
theta from layer 1 to 2

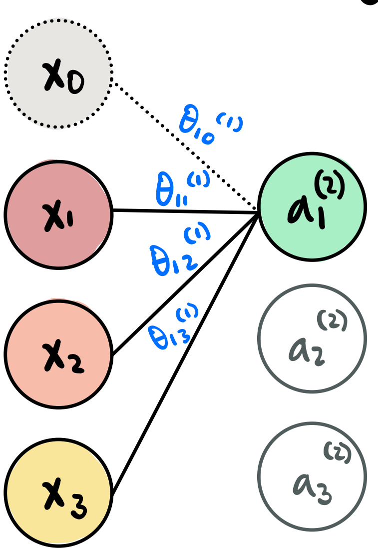
$$\theta^{(1)}_{30}$$

for the 0th node of the layer (1)

for the 3rd node of the next layer (2)

In matrix θ, each row will be the theta weights for a single neuron to multiply

$\theta =$ 



$$a_1^2 = g(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$

$$a_2^2 = g(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$

$$a_3^2 = g(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$

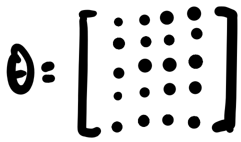
Notice
That these values determine what row in the theta matrix is used to calculate this neuron

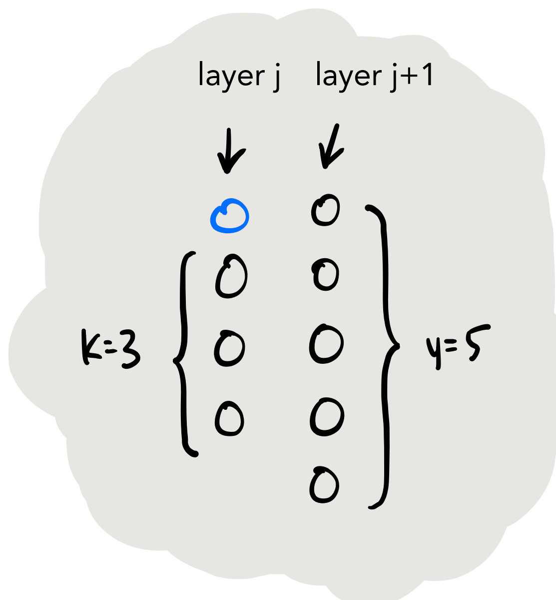
Q: What are the dimensions of theta of a single layer?

A: If a network has

- k units in layer j and
- y units in layer j+1,
- then Θ(j) has dimensions (y * k+1).
- Where 1 is added to k to adjust for the bias term

$\Theta: [\ ?\ ?\ ?\ ]$





Example:

layer 1 has 3 units and

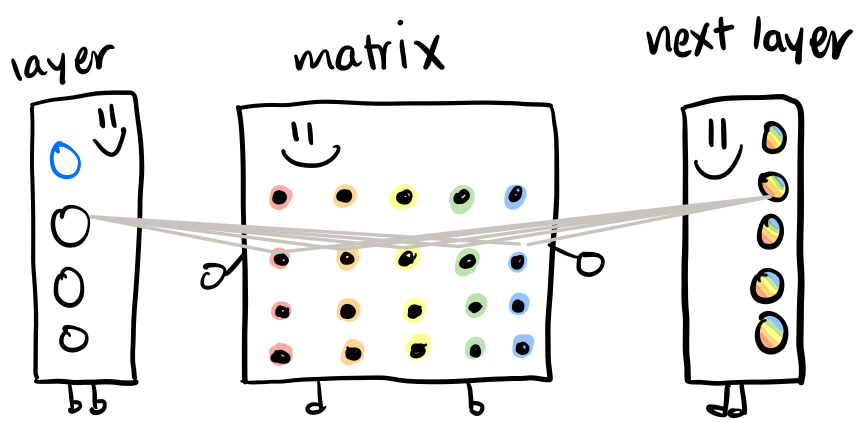
layer 2 has 5 units

$\theta^{(1)}$ will have dimensions:

(y * k+1) = (5 * 4)

dimensions:

(units in next layer x units in current layer plus bias unit)

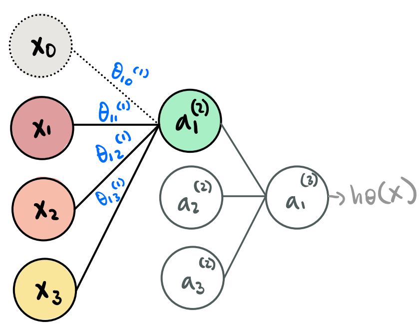


Model Representation 2

Q: How do we **calculate** the layers efficiently?

A: Computers are extremely fast at vectorized and matrix multiplication so we solved with vector math.

"Forward Propagation": Vectorized Implementation

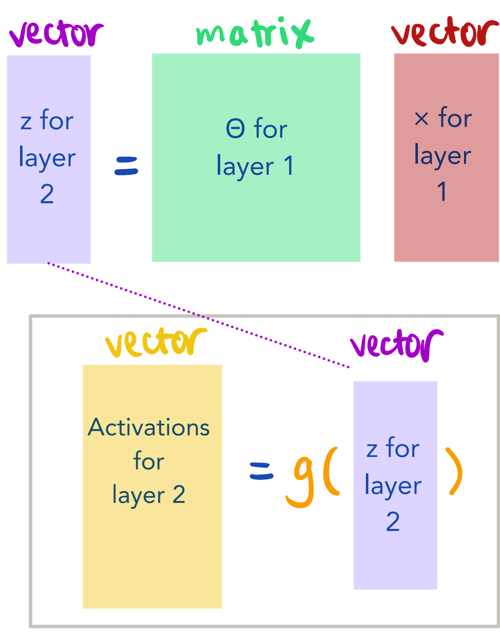


$$a_1^{2} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{2} = g({}_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{2} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} \ \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$



layer to

layer from

$$z^{(2)} = \Theta^{(1)} X$$

Z is the linear combination of theta and X values

layer

layer

$$a^{(2)} = g(z^{(2)})$$

$a^{(2)}$ is the activation layer of layer 2 that is input to layer 3

otherwise known as" layer 1''
or "activation layer 1"

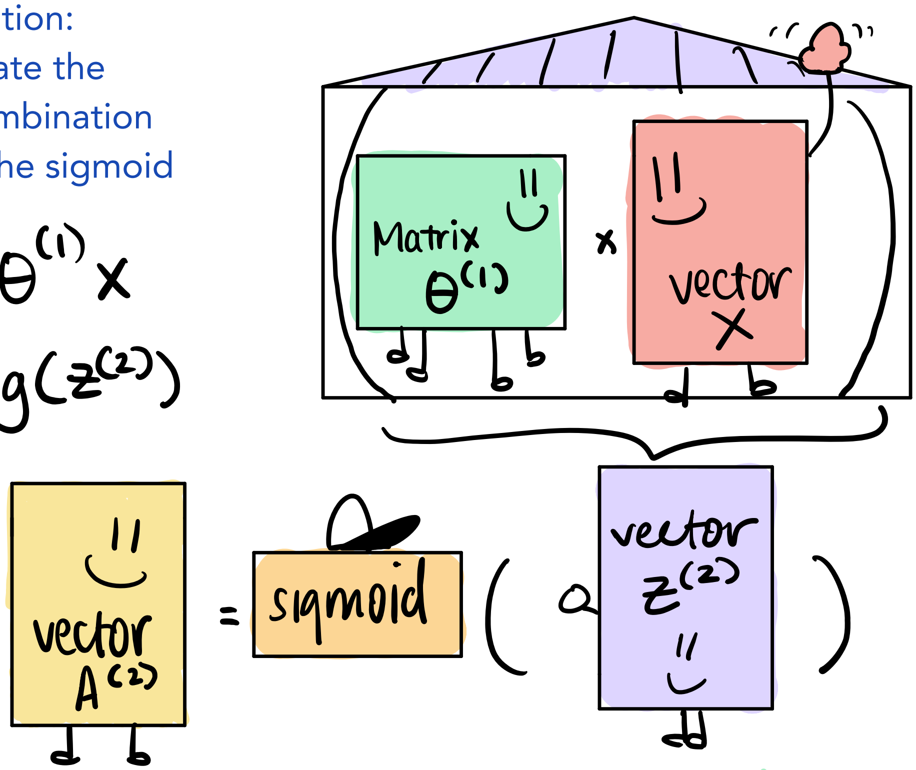
Say we have input vector x
Our first goal is to find activation layer 2.
Here we represent these vectors in shorthand

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \qquad a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

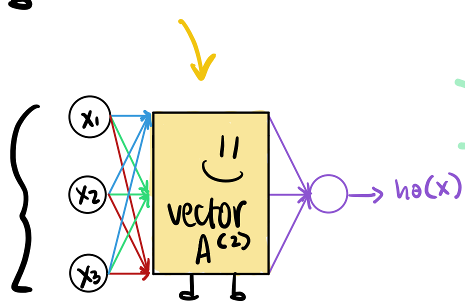
Solution:
calculate the
linear combination
and take the sigmoid

$$z^{(2)} = \Theta^{(1)} x$$

$$a^{(2)} = g(z^{(2)})$$

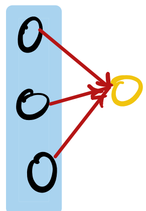


The input layer can also be called $a^{(1)}$ or activation layer 1



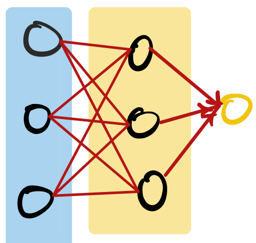
this process
"feeds forward"
into all further
layers

#NN learning its own features



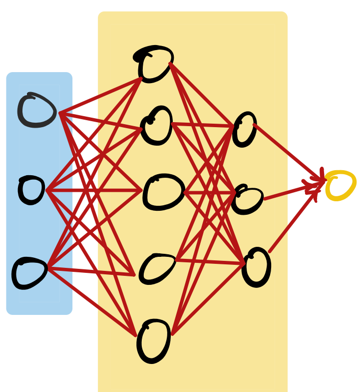
In logistic regression
these are the features
that contribute to output

one
more
layer



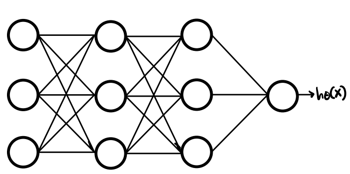
With neural networks
layers are added
that serve as inputs
to the next layer

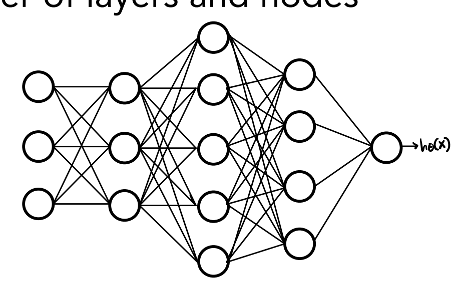
one
more
layer



Its like NN are
learning its own features
layer by layer until
the output

Note: we can choose different architectures
with a different number of layers and nodes



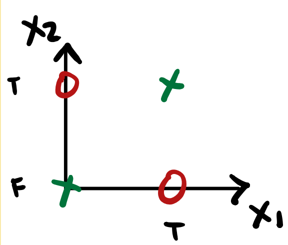


Neural Network: Examples and Intuitions

NN's allow us to model complex relationships that cannot be easily modeled as linear combinations

Two complex logical functions are
① XOR - exclusive OR
② XNOR - the inverse of XOR

What is XOR (exclusive OR)?

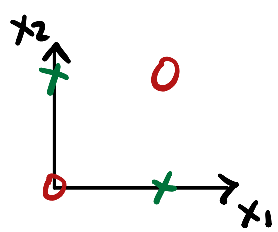


XOR returns true (T)
if x_1 OR x_2 are true
but not if
x_1 AND x_2 are true

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

note the inverse

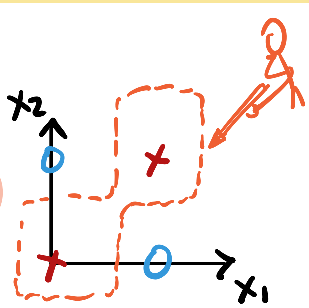
What is XNOR?



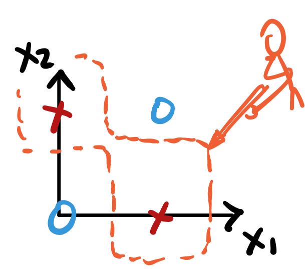
XNOR returns true
if x_1 AND x_2 are true
or if x_1 AND x_2 are false

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1

Imagine the difficulty in drawing a decision boundary



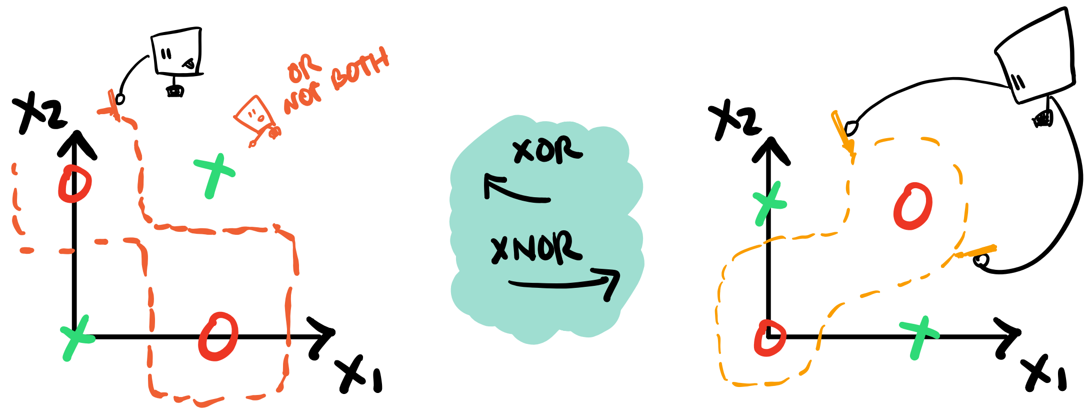
XNOR
XOR



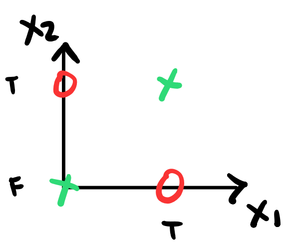
Neural Network: Examples and Intuitions

NN's allow us to model complex relationships
that cannot be easily modeled as linear combinations

Imagine how difficult it would be for a *linear function* to represent **XOR** (exclusive OR) or **XNOR** (inverse of XOR)



What is **XOR** (exclusive or)?

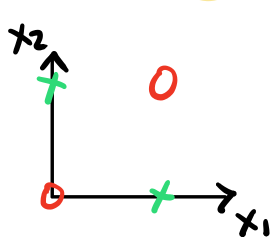


XOR returns true
if x_1 OR x_2 are true
but not if
x_1 AND x_2 are true

x_1	x_2	y
0	0	O
1	0	I
0	1	I
1	1	O

note the inverse

What is **XNOR**?



XNOR returns true
if x_1 AND x_2 are true
or if x_1 AND x_2 are false

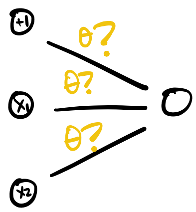
x_1	x_2	y
0	0	I
1	0	O
0	1	O
1	1	I

Q: How can we build XNOR?

Q: how do you build AND?

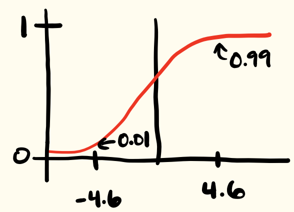
negative(-4.6) is .01

↓

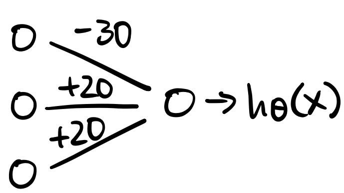


Q: What parameters (θ) would you need to create the AND function?

context: In our sigmoid function positive (+4.6) is .99 ↙ nearly 1 and negative (-4.6) is 0.01 ↙ nearly 0



Say we used parameters: -30, +20, +20

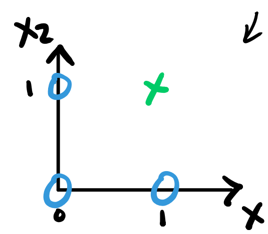


$= -30(1) + 20x_1 + 20x_2$

Truth Table

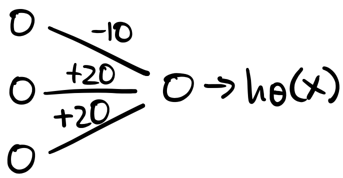
x_1	x_2	$h_\theta(x)$
0	0	0
1	0	0
0	1	0
1	1	1

we created this chart!



OR

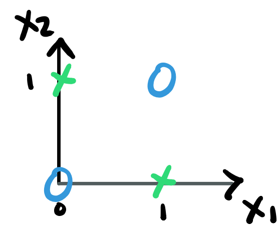
returns true
x_1 is true or x_2 is true



$= -10(1) + 20x_1 + 20x_2$

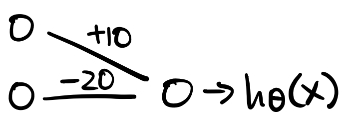
Truth Table

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1



NOT

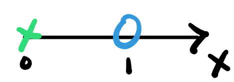
returns true
if x_1 is false



$= 10(1) - 20x_1$

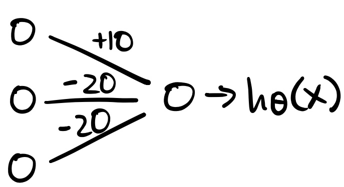
Truth Table

x	y
0	1
1	0



NOR

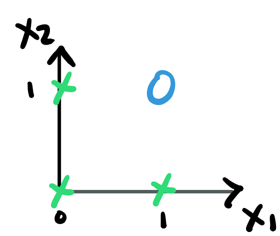
returns true
x_1 is false OR x_2 is false



$= 10(1) - 20x_1 - 20x_2$

Truth Table

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0



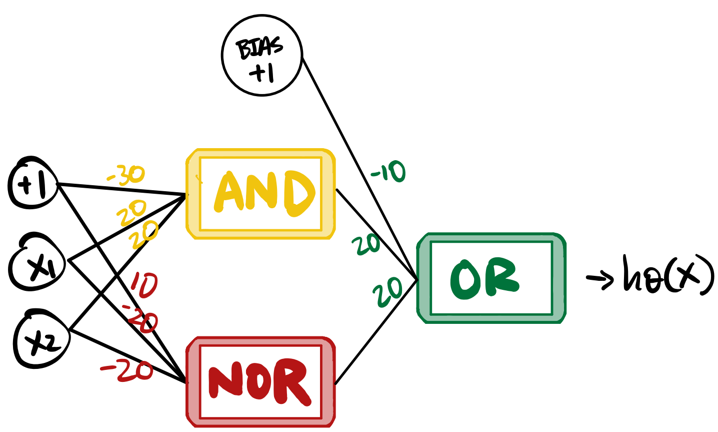
XNOR made of

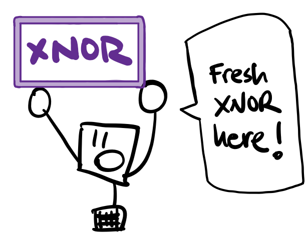
1. AND: x_1 and x_2
2. NOR: NOT x_1 and x_2
3. OR: x_1 OR x_2

AND $\theta^{(1)} = [-30, 20, 20]$

NOR $\theta^{(1)} = [10, -20, -20]$

OR $\theta^{(1)} = [-10, 20, 20]$



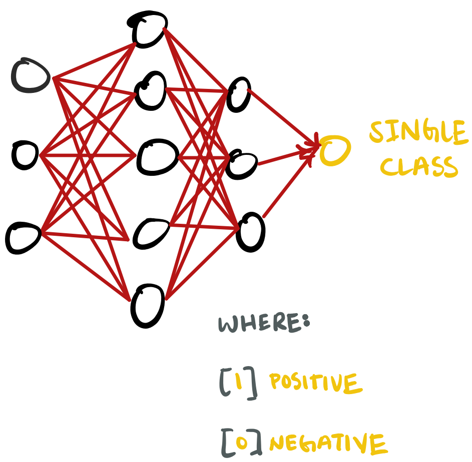


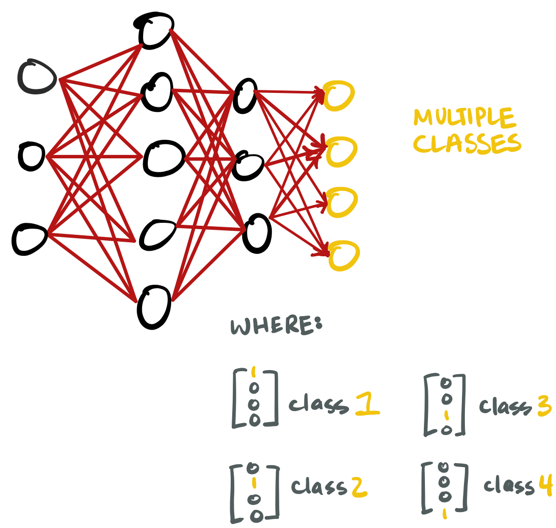
"Building" XNOR with AND, OR and NOT

$$\boxed{\text{XNOR}} = (x_1 \; \boxed{\text{OR}} \; x_2) \; \boxed{\text{AND}} \; \underbrace{(\boxed{\text{NOT}} \, (x_1 \; \boxed{\text{AND}} \; x_2))}_{\boxed{\text{NOR}}}$$

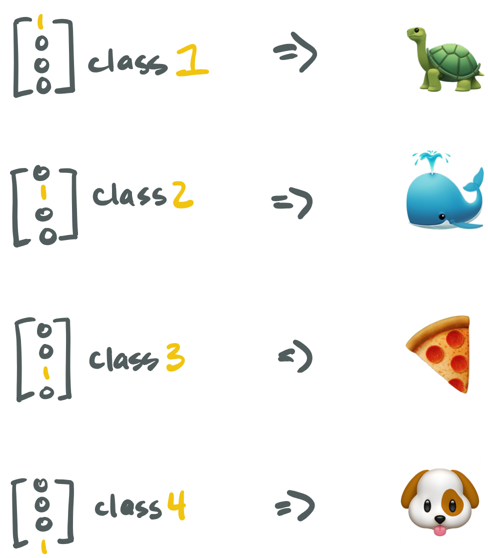
NN's can output to many nodes
to represent multiple classes

SINGLE CLASS vs MULTIPLE CLASSES





Multiple classes can be represented by one dimensional vectors where all values are 0 except for a single 1 value to represent the class



week5

week5

week5

week5

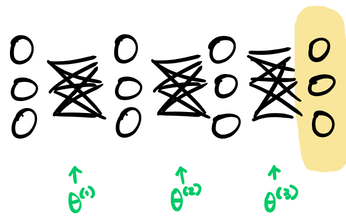
week5

wee

w

Neural Network Cost Function

Just like linear & logistic regression, we need a cost function,
the derivative of which will allow us to fit parameters
that will minimize cost



Terminology

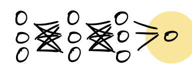
L : total no. of layers in network

s_l : no. of units in layer L

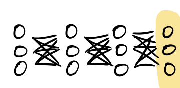
K : no. of output units

classification can be

- binary where K=1



- multi class where K >= 3



The NN cost function is a generalized version of
the logistic regression cost function adjusted for multiple
output nodes and theta dimensions of a matrix

cost function = $\boxed{\text{cost of predictions}}$ + $\boxed{\text{regularization}}$

Logistic regression cost function

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

with NN's there are K output nodes
so we have to sum over every K output

regularization now
accounts for layers
and multi dimensions
of theta matrix

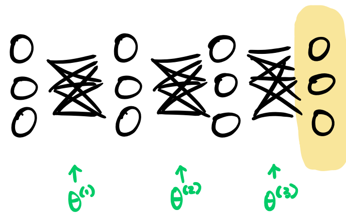
Neural network cost function

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log (h_\theta(x^{(i)}))_k + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

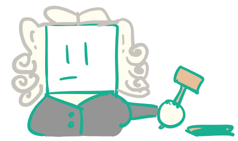
$$+ \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\theta_{ji}^{(l)})^2$$

Introducing backpropagation and why

Through the feed forward mechanism our NN creates an output prediction layer



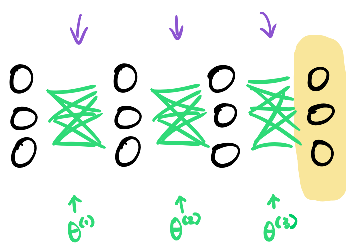
the cost function is like a judge



We have a cost function to evaluate our predictions.

Backpropagation: we calculate partial derivatives so we can nudge our theta (parameters/weights) by tiny amounts to minimize our cost "J(θ)"

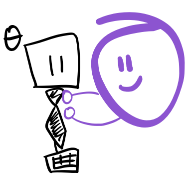
partial derivative update



Q: how do we find the partial derivatives?

Q: with some heavy math, we find that the partial derivative is the activation values multiplied by "delta" or the "error" term

δ



partial derivatives fixes the tie of theta

partial derivative	=	activations of layer	*	error term of next layer

partial derivative for a single training example

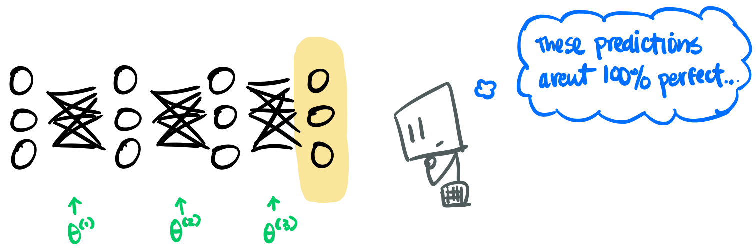
$$\frac{\partial J(\theta)}{\partial \theta^{(n)}} = a^{(n)} \delta^{(n+1)}$$

where δ is "delta" or the "error"
a is the activation values
n is the layer

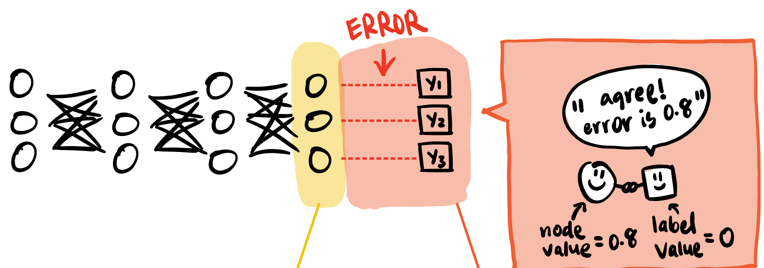
*we dont create the proof of this partial derivative in the lectures

Backpropation Algorithm

Through the feed forward mechanism our NN creates an output prediction layer

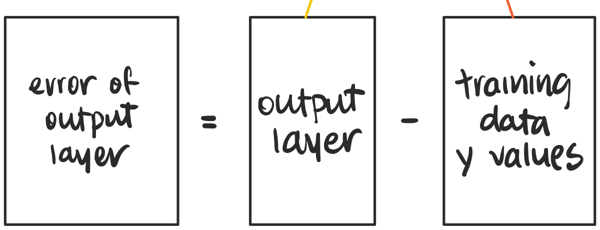


This output can be directly compared to the actual label values from the training set. The output layer **error terms** are straightforward

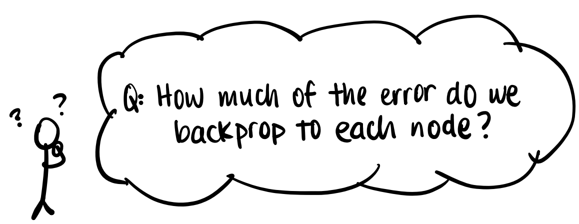


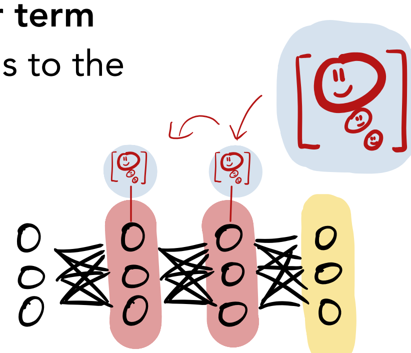
$$\delta^{(L)} = a^{(L)} - y$$

where L is the output layer



Q: How can we use the output layer **error term** ("delta") to <u>propagate back</u> error terms to the other layers?





Calculating delta for layers

output layer delta = output layer - y labels

$$\delta^{(K)} = a^{(K)} - y$$

hidden layer delta is a function of θ, delta of the next layer and derivative of sigmoid(z) and values

$$\delta^{(n)} = (\Theta^{(n)})^T \delta^{(n+1)} \; .* \; g'(z^{(n)})$$

where $g'(z^{(n)}) = a^{(n)} \; .* \; (1 - a^{(n)})$

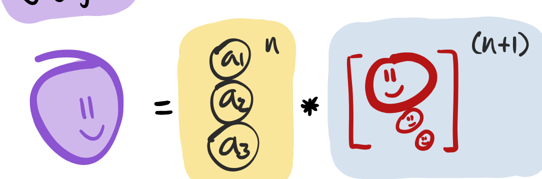
fully expanded:

$$\delta^{(n)} = (\Theta^{(n)})^T \delta^{(n+1)} \; .* \; (a^{(n)} \; .* \; (1 - a^{(n)}))$$

These delta values are used in partial derivative to determine changes in theta "parameters"

single example →

$$\frac{\partial J(\theta)}{\partial \theta_{i,j}^{(n)}} = a_j^{(n)} \delta_i^{(n+1)}$$



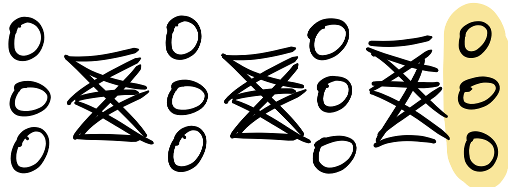
For our training set, the partial derivatives are averaged where m is the total number of training examples t

multiple example →

$$\frac{\partial J(\theta)}{\partial \theta_{i,j}^{(n)}} = \frac{1}{m} \sum_{t=1}^{m} a_j^{(t)(n)} \delta_i^{(t)(n+1)}$$

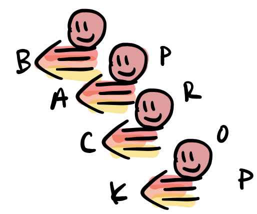
* note: for simplicity we leave out the regularization term

Forward propagation calculates activation nodes and the output layer



$a^{(1)}$ $\theta^{(1)}$ $a^{(2)}$ $\theta^{(2)}$ $a^{(3)}$ $\theta^{(3)}$ $a^{(4)}$

the output layer calculates the delta error term and **backpropagate** error terms for every layer



$$\delta^{(4)} = a^{(4)} - y$$

$$\delta^{(3)} = (\theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

theta used to compute next layer

delta of next layer

dot product

sigmoid derivative $= a^{(n)} .* (1 - a^{(n)})$

Input layer error is not calculated

$a^{(1)}$ $\theta^{(1)}$ $a^{(2)}$ $\theta^{(2)}$ $a^{(3)}$ $\theta^{(3)}$ $a^{(4)}$

error terms are used to calculate the **partial derivatives**

$$\frac{\partial J(\theta)}{\partial \theta^{(3)}_{ij}} = \boxed{a_j^{(3)} \delta_i^{(4)}}$$

this multiplication creates a matrix that has dimensions for the matrix $\theta^{(3)}$

$$\frac{\partial J(\theta)}{\partial \theta^{(2)}_{ij}} = a_j^{(2)} \delta_i^{(3)}$$

$$\frac{\partial J(\theta)}{\partial \theta^{(1)}_{ij}} = a_j^{(1)} \delta_i^{(2)}$$

* this notation is for one training example. The actual training set will average over all training examples

complete notation	$\frac{\partial J(\Theta)}{\partial \Theta^{(l)}_{i,j}} = \frac{1}{m} \sum_{t=1}^{m} a_j^{(t)(l)} \delta_i^{(t)(l+1)}$

“If this is difficult,
you are not alone”

"I’ve actually used back propagation
pretty successfully for many years
and even today I still don’t, sometimes,
feel like I have a very good sense
of just what it's doing or sort of intuition
about what back propagation is doing"

- Andrew Ng