

Machine Learning

- Andrew Ng

Week 1:
introduction

*illustrated by
Patricke*

What is machine learning?

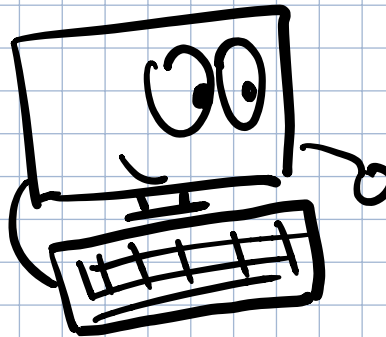
OLDER
INFORMAL
DEFINITION

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

- Arthur Samuel

Computer
Program:
Explicitly Programmed

This computer has been
programmed to identify
a bear



Instructions:

while plugged in,
look at the pixel 360
to pixel 368,
if these are brown
then return "bear"



What is machine learning?

MORE MODERN DEFINITION



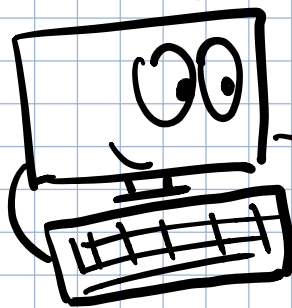
A computer program is said to learn from experience E with respect to some class of tasks T and performance P

IF

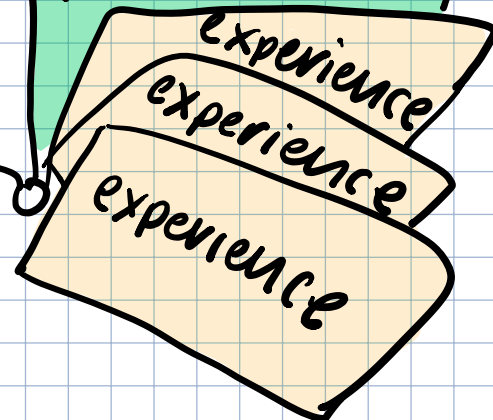
its performance at tasks T
as measured by P
improves with experience E
- Tom Mitchell

Experience $\Rightarrow$ Improved
Performance
@Tasks

Computer
Program
Learning:



way to learn
from experience

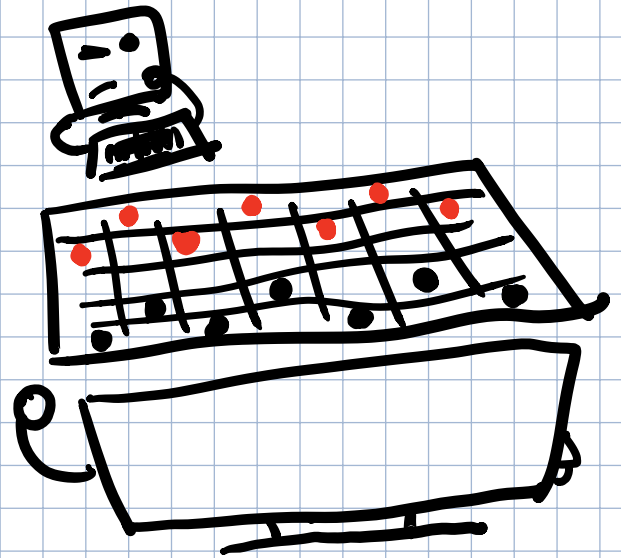


Example: Computer Program playing checkers

E (experience): playing many games of checkers (or data of games)

T (tasks) : task of playing checkers

P (performance): probability of winning



This course covers supervised & unsupervised learning

Supervised Learning

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

housing data & house prices

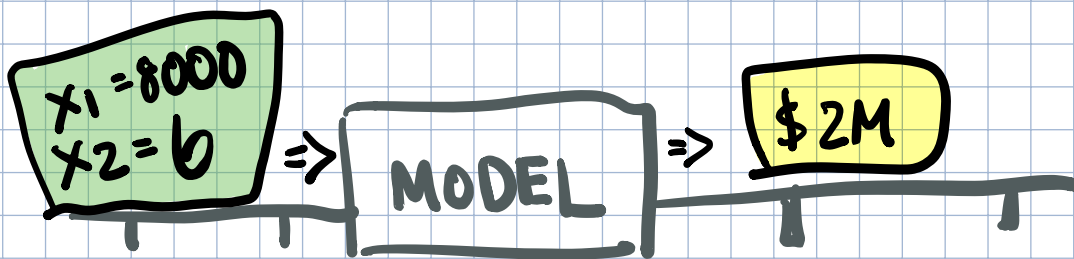
This data is used to create a model
use to predict new data

model (new housing data) $\Rightarrow$ predicted house price

Data		
x_1 size	x_2 bedrooms	y price
1000	1	\$100K
2000	3	\$800K
1500	2	\$700K
$\vdots$	$\vdots$	$\vdots$

$\uparrow$ $\uparrow$ $\uparrow$
these are inputs output

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



two main flavors 🍦🍦

Supervised learning

linear regression

predict results in a continuous space

Example 1

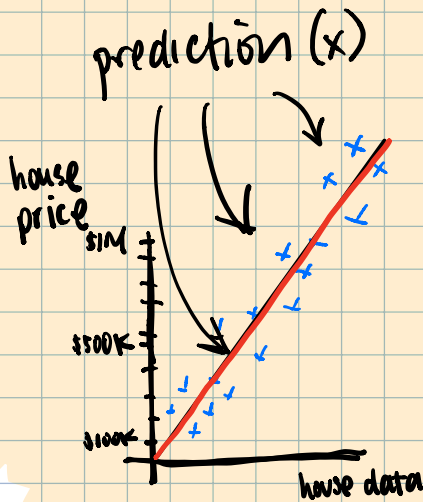
given: house data

predicts: house price

Example 2

given: photo of a person

predicts: age of person



classification

predicts results in a discrete space

Example 1

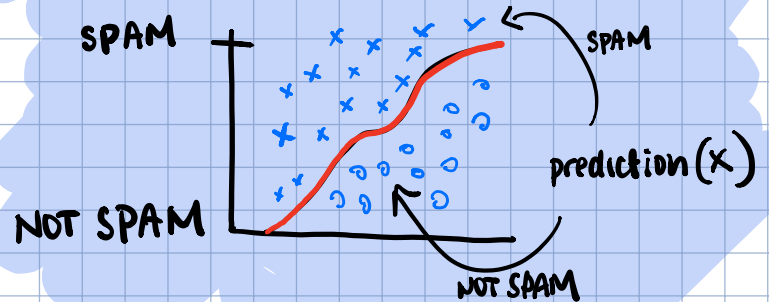
given: email data

predicts: spam or not spam

Example 2

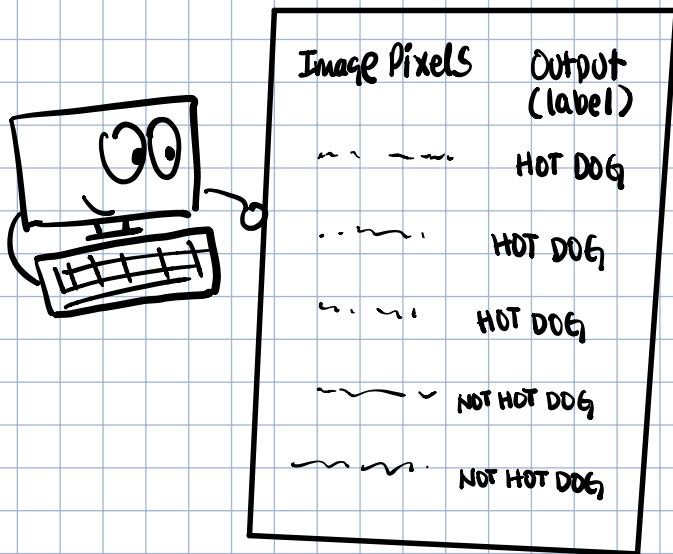
given: photo of tumor

predicts: malignant or benign



🌸 ART IN LIFE 🌸

HBO's Silicon Valley created a real app called "Not Hot Dog". They used a total of 150,000 images (3,000 hot dog images) to train their model to identify all types of hot dogs.



↑
This is a great example of logistic regression where image is the input {"HOT DOG", "NOT HOT DOG"} is output

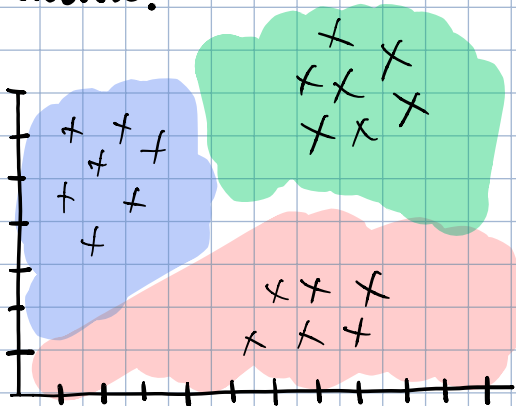
two main flavors 🍦🍦

Unsupervised Learning

(outputs are not in our provided data)

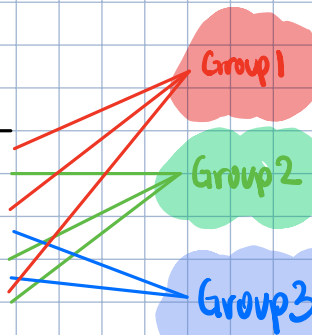
clustering

"find clusters!"



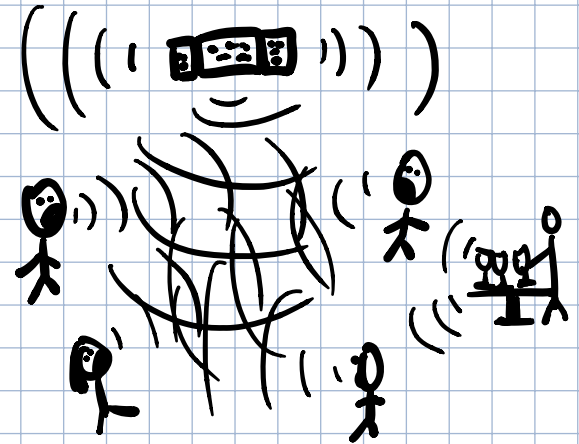
Say we are given gene data, (location, lifespan, role) that we would like to group

id	location	lifespan	role
1	⋮	⋮	⋮
2	⋮	⋮	⋮
3	⋮	⋮	⋮
4	⋮	⋮	⋮
5	⋮	⋮	⋮
6	⋮	⋮	⋮



not clustering

example: Cocktail Party Problem



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

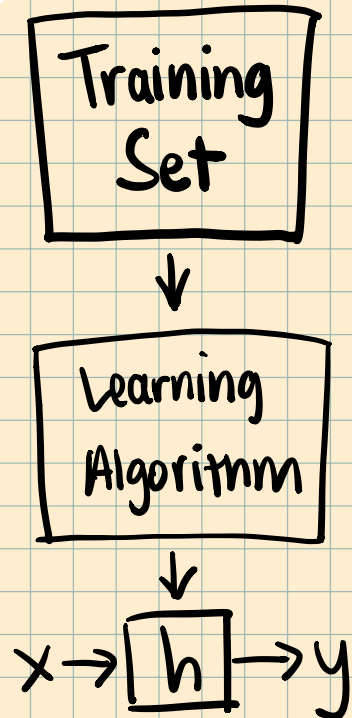
Supervised Learning

We are given a set of data
made up of inputs and outputs
Our goal is to learn a function

$$h(x) \rightarrow y$$

Hypothesis function (h)

given input (x)
predicts output (y)



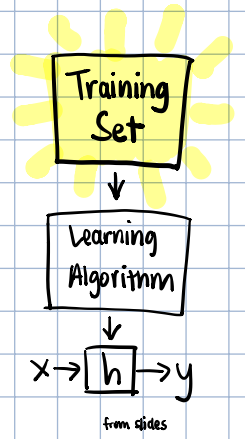
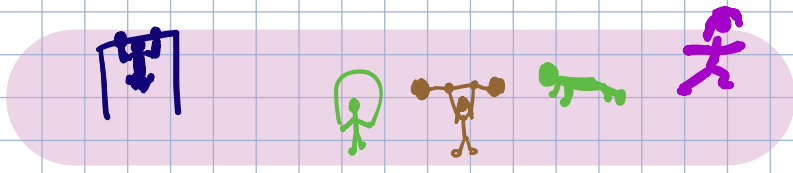
from slides

Note: the heavy lifting occurs in the learning algorithm. You can think of this as a lab or factory that builds and refines the hypothesis function



Training Set

Our data is called our training set



Q: what makes up our data?

In supervised learning, our data has:

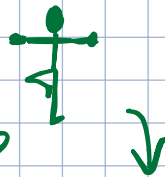
- ① inputs, $x^{(i)}$ (our "features")
- ② outputs, $y^{(i)}$ (our "labels")

For example...

the inputs are

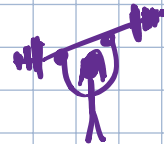
$x_1 = \text{size}$

$x_2 = \text{bedrooms}$



output is

house price



inputs			output
i	x_1 size	x_2 bedrooms	y \$ price
1	1000	1	\$100K
2	2000	3	\$800K
3	1500	2	\$600K
⋮	⋮	⋮	⋮

There could be 100's, 1000's, even 1,000,000's of input variables!

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

$m = \text{number of training examples}$

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

NOTE: inputs are eventually converted to numeric inputs, even images!

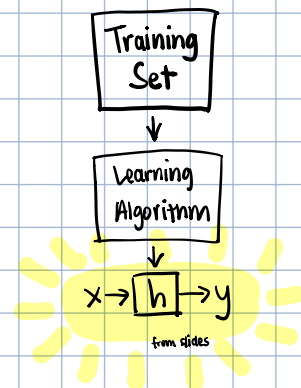
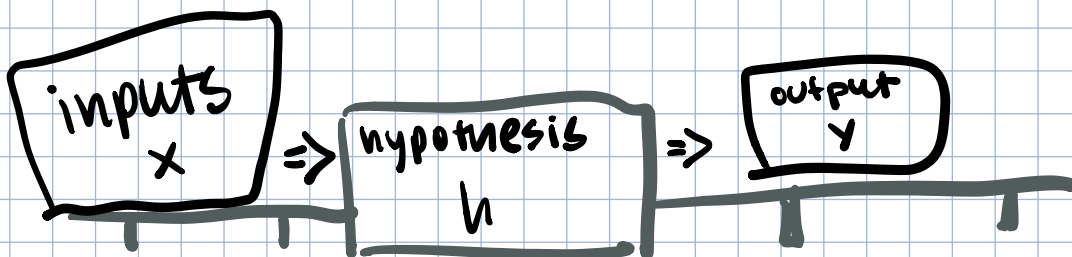


$\Rightarrow$ 01010110
10111000
101100...
...

HYPOTHESIS FUNCTION (h)

Q: What does the hypothesis function (h) look like?

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

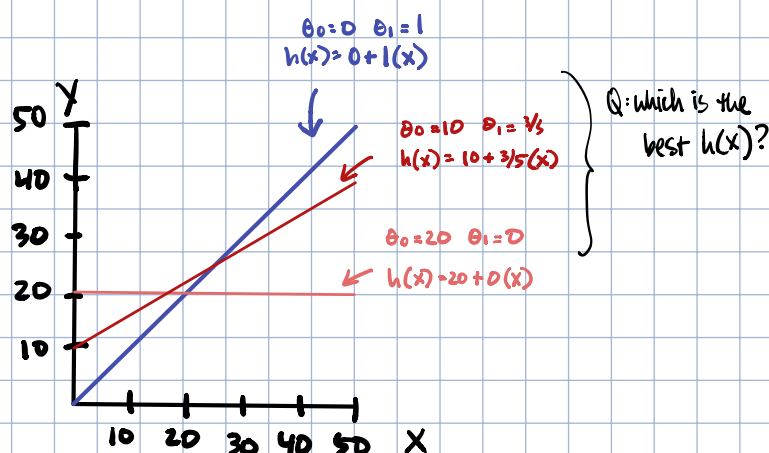


Example h for data with one input (x)

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

↑ ↑

we want to solve for θ_0 & θ_1 , so that $h(x)$ "fits" well with our training set



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

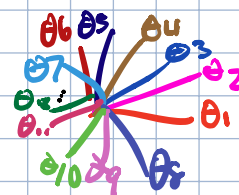
$$2 \Rightarrow h\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

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

⋮

$$100 \Rightarrow h\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5 + \theta_6 x_6 \dots$$

↑ this many dimensions becomes hard to graph so the course usually visualizes less features



8 = 3

I've noticed $h, h_0, h_0(x)$ being used... ahh
These all refer to the hypothesis function
where h_0 more specifically states that h is
parameterized by θ , $h_0(x)$ says
it takes input (x)

Note: depending on our domain knowledge about the data and the problem we can create complex hypothesis'

Example: two inputs with a special relationship

$$h_0 = \theta_0 + \theta_1 x_1 + \theta_2 x_1 x_2 + \theta_3 x_2$$

Example: two inputs and higher order relationships

$$h_0 = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$$

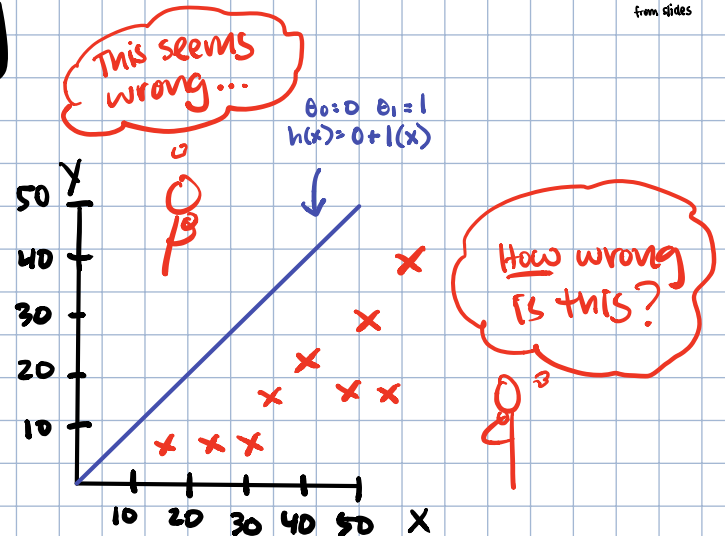
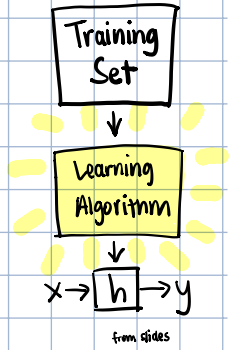
"whichever function we use will depend on our knowledge about the data"

~~~~~ cost function ~~~~~

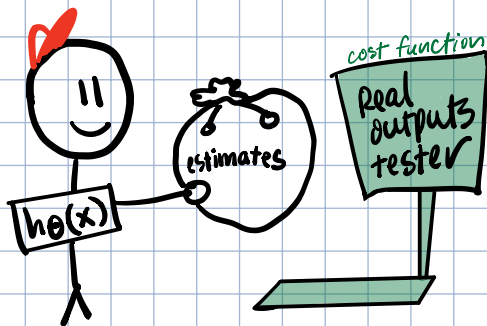
Q: How do we evaluate our hypothesis function $h(x)$?

Q: First, why do we have to evaluate h ?

A: As we solve for θ ,
we need to know how wrong
our estimate for θ is
so we know whether
or not to adjust θ



A: Remember, with supervised learning our data includes the correct outputs we can use this to compare our $h(x)$ outputs with the real outputs. There are different theories about the best cost function but a common one uses average squared difference.



Linear Regression Cost Function

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

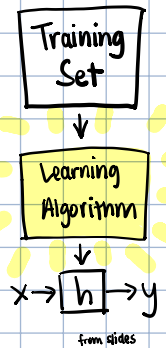
↑ ↑ ↑ ↑
averaging factor sum difference square

Note: this function amplifies errors where a diff of 1 is $1^2=1$ while a diff of 5 is $5^2=25$!

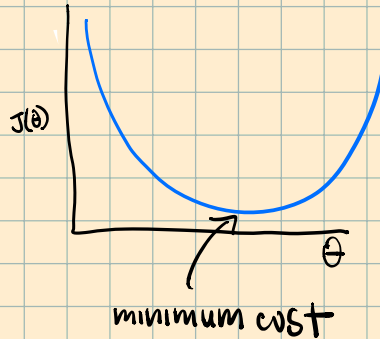
Gradient Descent

Q: What is gradient descent (GD)?

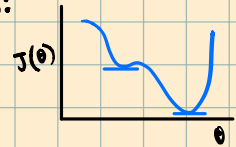
A: GD is an algorithm that allows us to find the θ (meta) values that lead to the lowest $J(\theta)$ (cost). * note: later, we discuss local minima



The cost function for linear regression could look something like this ->



Note: linear regression will always be convex and doesn't have local optima like this:

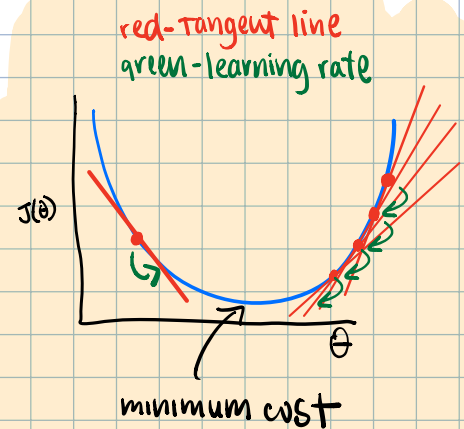


Q: How?

A: The derivative of the cost function informs us of the slope of the tangent line. With this we know which direction to "descend".

A "learning rate" we choose determines how large the step sizes are.

Remember calculus?
The derivative gives us the tangent line



Gradient Descent Algorithm

(one input example)

This is the condensed algo for one input and two theta's

learning rate $\downarrow$ partial derivative

repeat {

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

} $\uparrow$ perform simultaneously for all j

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

This is the expanded partial derivative from calculus

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

* 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 {

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

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

} $\uparrow$ update simultaneously

week 2

Multiple variables

Our data may contain

a single input feature.

here: size

x	y
Size	price
1000	\$100K
2000	\$200K
3000	\$300K
⋮	⋮

or

multiple input features

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

x_1	x_2	x_3	x_4	y
Size	# bedrooms	# floors	age (years)	price
1000	1	1	60	\$100K
2000	2	1	15	\$200K
3000	3	2	48	\$300K
⋮	⋮	⋮	⋮	⋮

$x^{(2)}$ → (points to the row for 2000 size, 2 bedrooms)

$x_2^{(3)}$ → (points to the value 3 in the # bedrooms column)

Notation

n = number of features

$x^{(i)}$ = i th training example

$x_j^{(i)}$ = i th training examples j th feature

\$???



← size?

← bedrooms?

← floors?

← age?

Hypothesis Function

Multi-variate + Vectorized

Single feature hypothesis function

$$h_0(x) = \theta_0 + \theta_1 x$$

Multiple Variable hypothesis function

$$h_0(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 \quad \leftarrow \text{(four features)}$$

or generally

$$h_0(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

Note: we have more θ values than x values because of the θ_0 bias value

In order to create a vectorized representation of $h_0(x)$ we can assume an x_0 value (which is always 1)

Now our vectors both have length $n+1$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Multiplying these two vectors

$$h_0(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

can be represented as

Multiple feature hypothesis function

$$h_0(x) = \theta^T x$$

why is $x_0 = 1$?

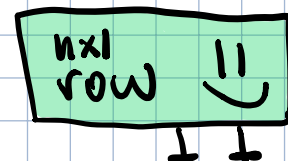


x_0 is just a placeholder and 1 times any value is that same value

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



+ Transpose



for example

$$\theta^T = [\theta_0 \ \theta_1 \ \theta_2 \ \dots \ \theta_n]$$

Gradient Descent

Gradient Descent works similar going from one feature to multiple features

(n=1) single feature

repeat {

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

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

}

(simultaneously update θ_0, θ_1)

(n > 1) multiple features

repeat {

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

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

}

...

⇓

more generally

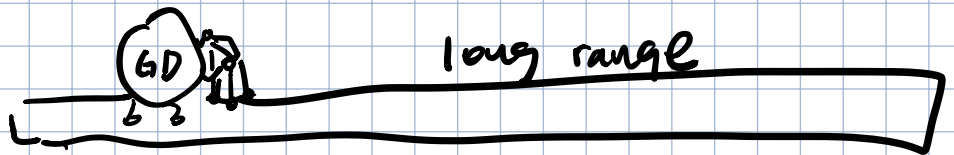
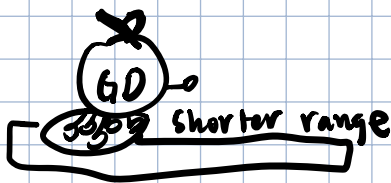
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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

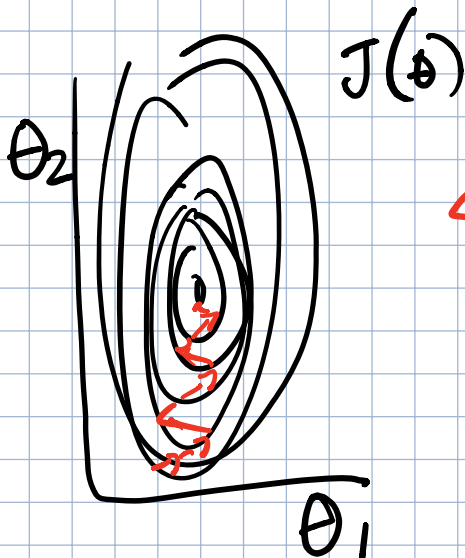


Say you have two features

x_1 = size (0-2000 feet)

x_2 = # of bedrooms (1-5)

The difference in scale creates a skewed cost curve graph



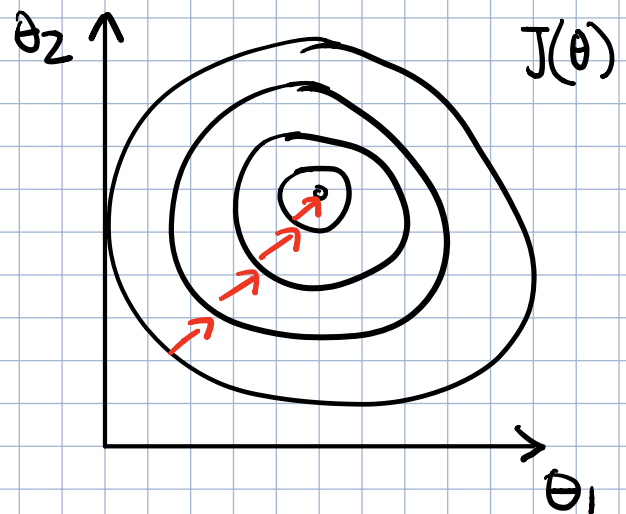
← gradient descent may take super long

Feature scaling: divides input values by its range

$$x_1 = \frac{\text{size}}{2000}$$

$$x_2 = \frac{\text{\# of bedrooms}}{2}$$

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 $-3 \leq x \leq 3$ and $-\frac{1}{3} \leq x \leq \frac{1}{3}$ 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

In the above example if:

average size = 1000

average bedrooms = 2

$$x_1 = \frac{\text{size} - 1000}{2000}$$

$$x_2 = \frac{\# \text{ bedrooms} - 2}{5}$$

combined:

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

new feature value

old feature value

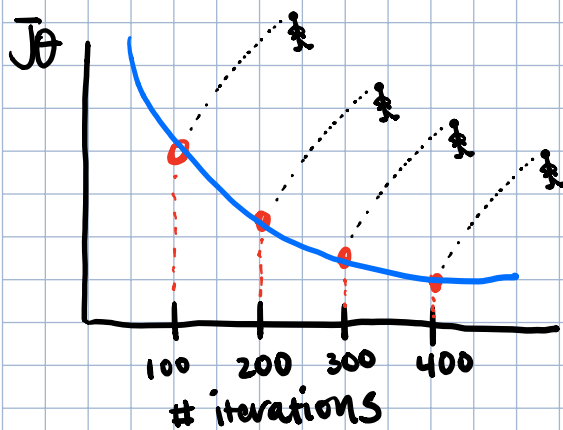
mean

range

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

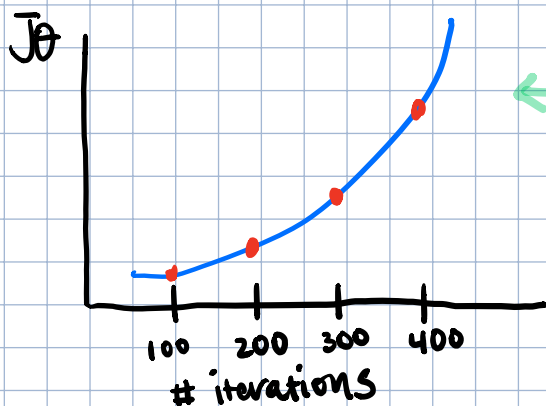
Sometimes GD never converges
Sometimes GD has a slow convergence

Method 1: plot the cost function $J(\theta)$ to its number of iterations



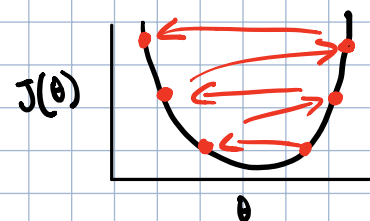
Use checkpoints, here 100, 200, 300, 400 iterations to see if gradient descent is working properly.
 $J(\theta)$ should decrease with every iteration

what if our graph looked like this?



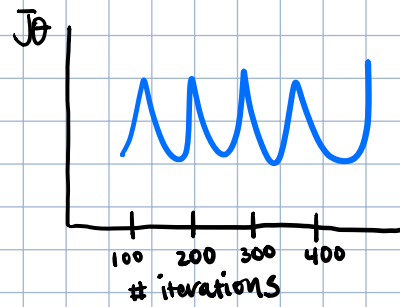
Solution:
Use a smaller learning rate α

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



the learning rate α is causing our us to overshoot the minimum and actually increasing $J(\theta)$

After plotting, our graph could also look like this →
In this case, try using a smaller learning rate α .



Learning Rate (alpha α)

small α

- for sufficiently small α , $J(\theta)$ should decrease on every iteration
- BUT if α is too small, gradient descent can be slow to converge

LARGE α

- if α is too large, $J(\theta)$ may not decrease on every iteration AND it may never converge

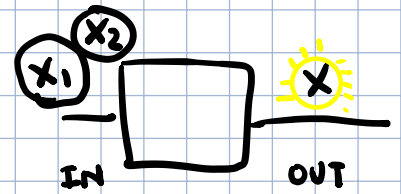
To choose an α try different values:

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

 ↗
 $\approx \times 3$ ↗
 $\times 3$ ↗
 $\times 3$ ↗
 $\times 3$

FEATURES

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



Given two features

$x_1 = \text{frontage}$

$x_2 = \text{depth}$

Housing prices prediction

$$h_{\theta}(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2}$$



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

$$\text{area} = \text{frontage} \times \text{depth}$$

Now we can use a better predicting single feature $h_{\theta}(x)$

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

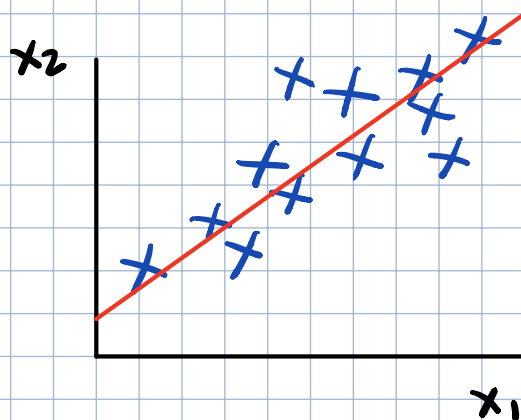
POLYNOMIAL REGRESSION

We have been working with straight lines so far

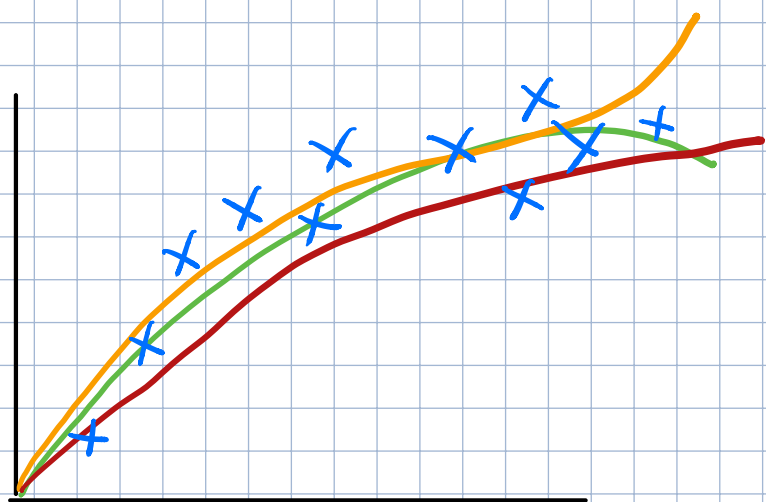
$$h\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

↑

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



say $x = \text{size}$

quadratic

$$\theta_0 + \theta_1 x + \theta_2 x^2$$

OR

cubic

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

OR

square root!

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

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

if $\text{size} = 1-1000$

$$\text{size}^2 = 1-1,000,000$$

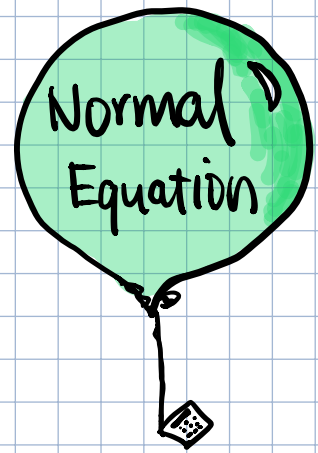
$$\text{size}^3 = 1-1,000,000,000$$

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 1: Normal Equation

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



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

where x is our inputs
 y is our output

How:

$$\text{Normal Equation: } (X^T X)^{-1} X^T y$$

where X = inputs

y = outputs

m = # training examples

n = # features

10:51 AM Sat Dec 7

Normal Equation

Examples: $m = 4$.

	Size (feet ²)	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

$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$
 $m \times (n+1)$

$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$
 m -dimensional vector

$\theta = (X^T X)^{-1} X^T y$

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

HOW IN OCTAVE:

$$\text{pinv}(X' * X) * X' * y$$

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 $(X^T X)^{-1}$ - Slow if n is very large

Lecture 2: Normal Equation and Non-invertibility (optional)

Q: what if $X^T X$ is non invertible?

A: This should happen very rarely
but this may be 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 A: delete features

Week 3

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, 0 or 1

Given features x , $h_\theta(x) = \{0, 1\}$



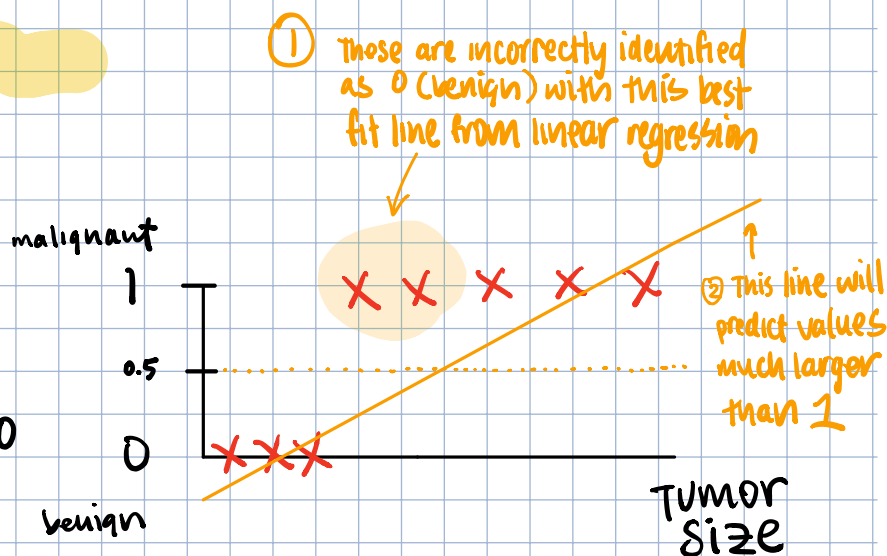
Note: $\{1, 0\}$ is binary and can be interpreted as $\{\text{Yes}, \text{No}\}$, $\{\text{SPAM}, \text{NOT SPAM}\}$, $\{x, 0\}$, $\{\text{HOT DOG}, \text{NOT HOT DOG}\}$

Looking at a line through data

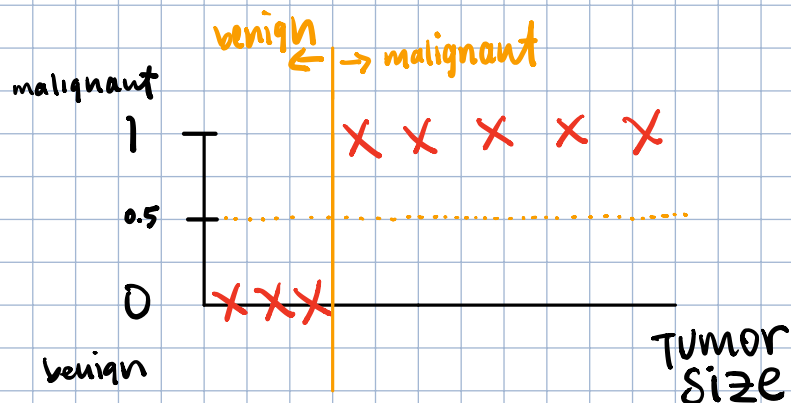
Here we use the best fit line from linear regression:

We may be tempted to interpret

$h_\theta(x) > 0.5 = 1$ and $h_\theta(x) < 0.5 = 0$



We want to go from best fit to best split



Intuition:

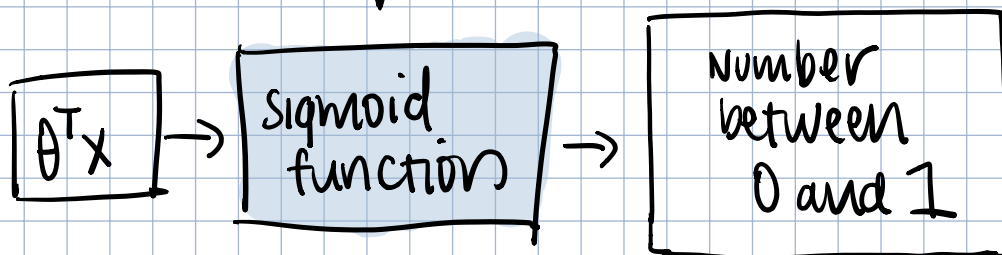
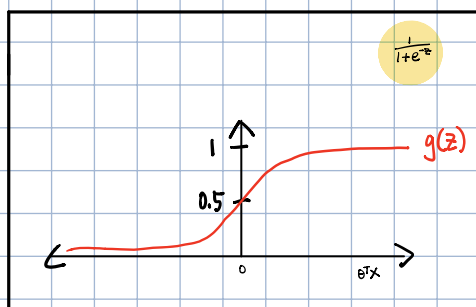
This was our
linear regression
hypothesis function

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



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



Logistic Regression Hypothesis Function

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

where

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

one liner:

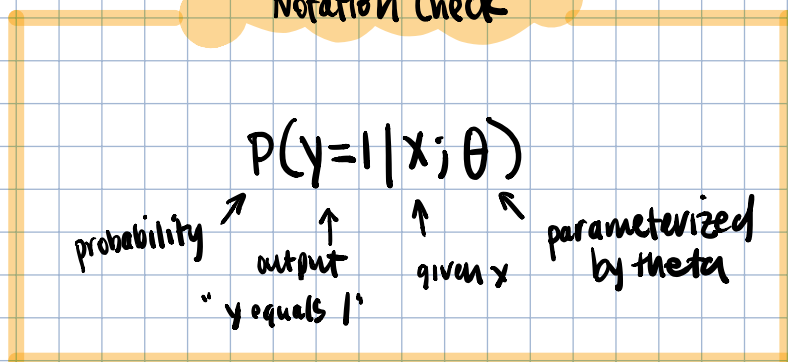
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$



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



More about probability algebra

There is a 100% probability that $y=1$ OR $y=0$

Therefore:

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

$$P(\text{DO}) + P(\text{DO NOT}) = 1$$

$$P(\text{SPAM}) + P(\text{NOT SPAM}) = 1$$

$$P(\text{HOT DOG}) + P(\text{NOT HOT DOG}) = 1$$

$$P(\text{NOT HOT DOG}) = 1 - P(\text{HOT DOG})$$

Note: once you know

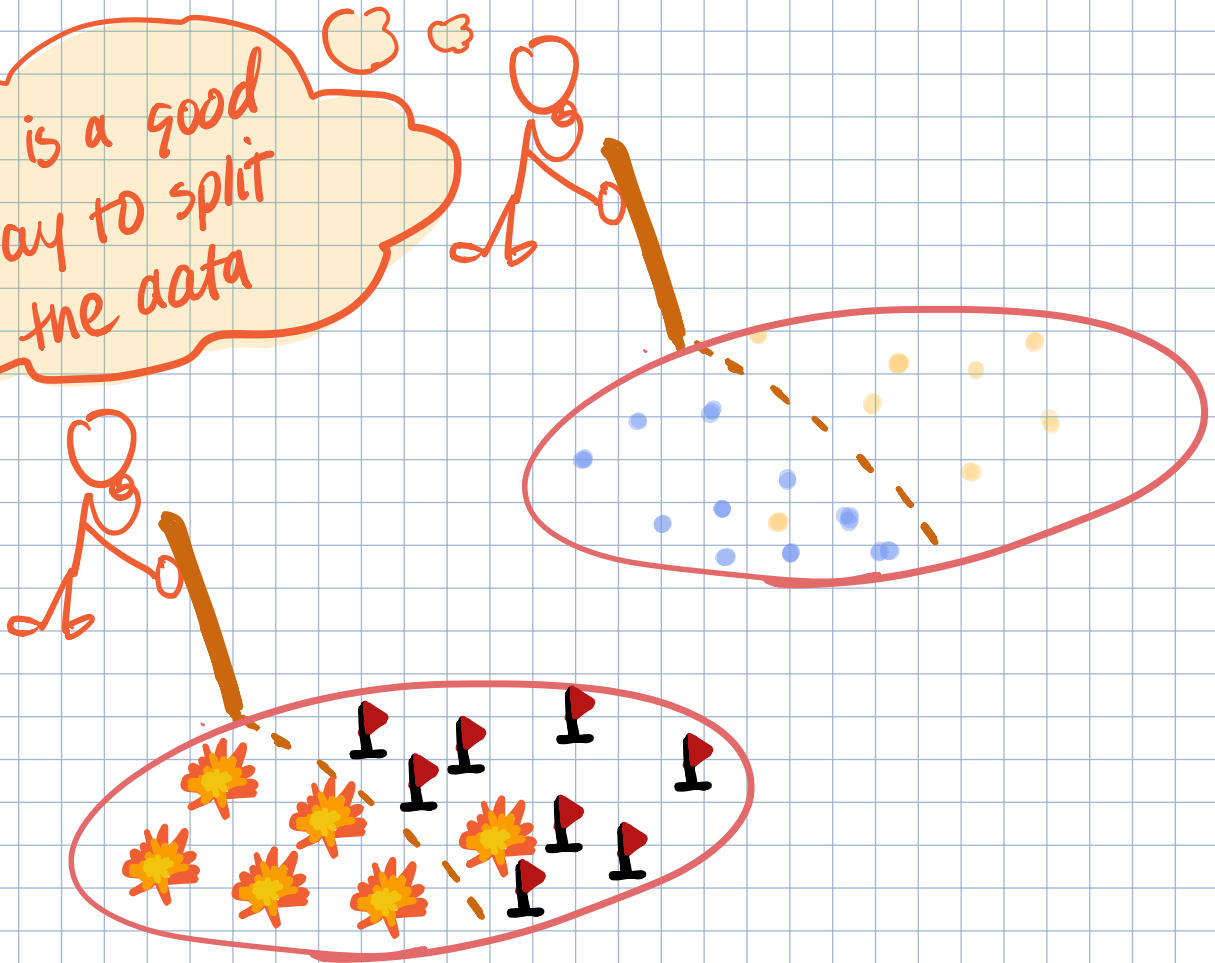
$P(y=1)$ or $P(y=0)$ you

can derive the other through algebra!

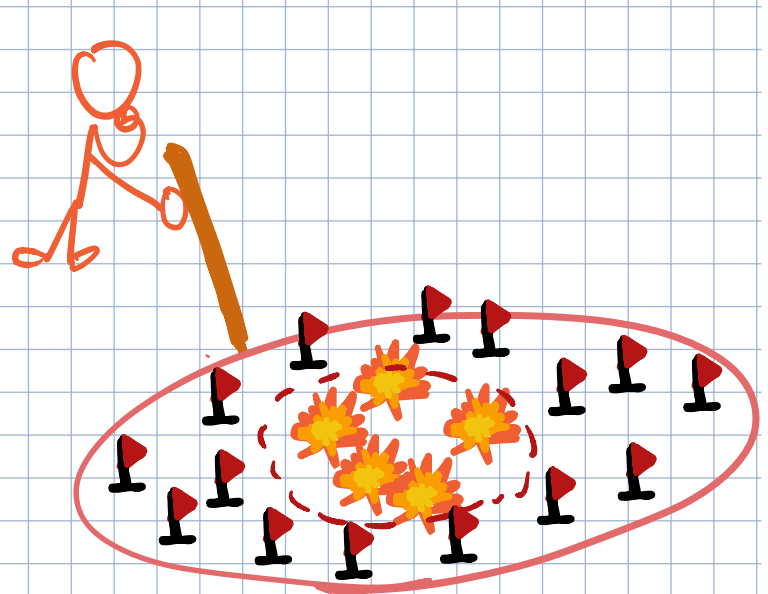
Decision Boundary

Finding the boundary to best split the data

This is a good way to split the data



This data is better split with a circle



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

Say we have...

this hypothesis function and this data

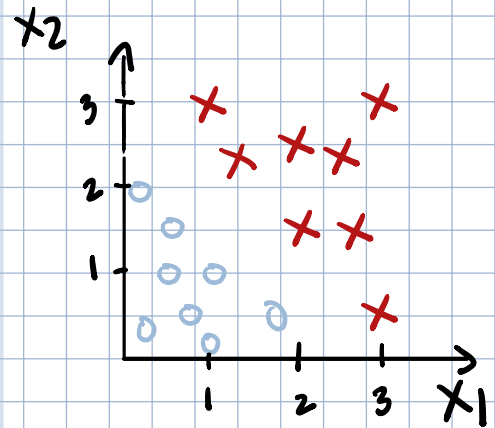
Hypothesis Fn.

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

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:

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

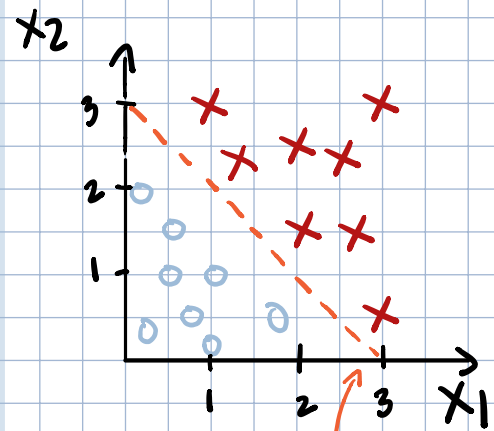
Hypothesis Fn.

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

and predict $y=1$

if $h_{\theta}(x) \geq 0$

Charted Data



This is the decision boundary solved by finding theta

LOGISTIC REGRESSION COST FUNCTION

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

A: NO

Q: Why not?

First let's look at our linear regression cost function:

$$\text{linear regression } J\theta = \frac{1}{n} \sum_{i=1}^n \underbrace{\frac{1}{2} (h\theta(x^{(i)}) - y^{(i)})^2}_{\text{non-linear}}$$

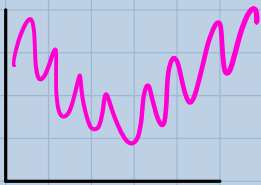
A: Since we have a higher order (non-linear)

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

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

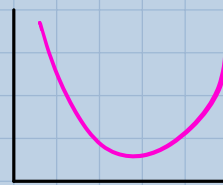
NON-CONVEX

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



CONVEX

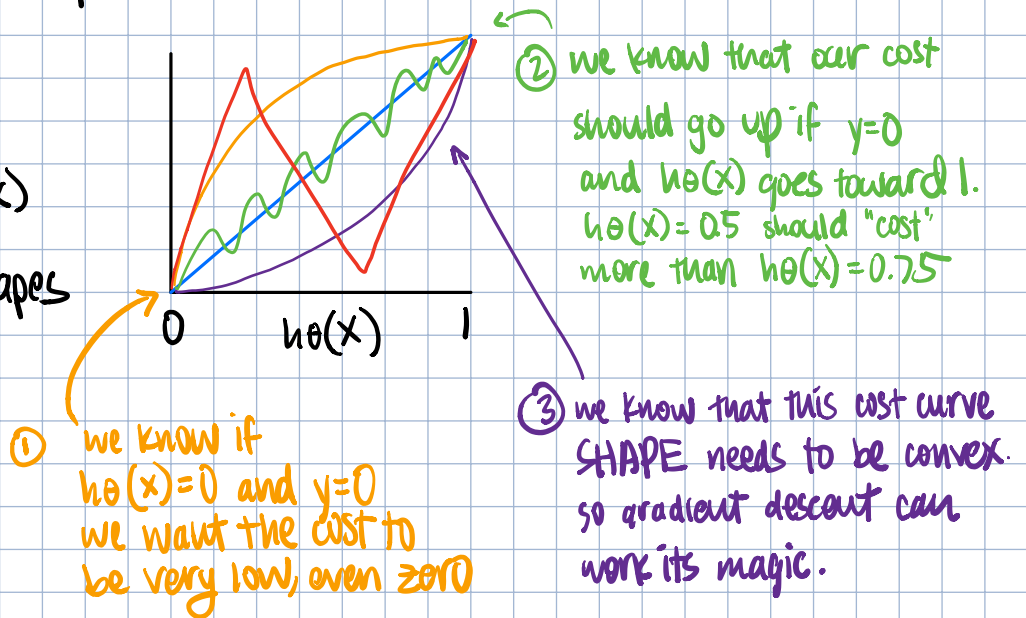
Our goal is to get this pretty looking convex shape that gradient descent can help with



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

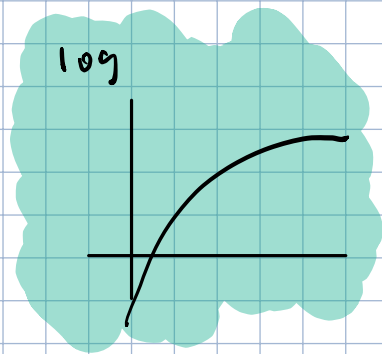
if $y=0$

our cost graph for $h_\theta(x)$
could be any of these shapes

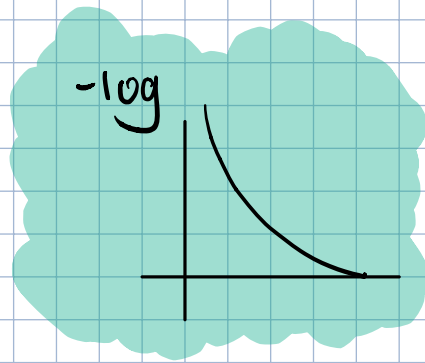


A: The log and $-\log$ shapes are great shapes

This is the
log function graph

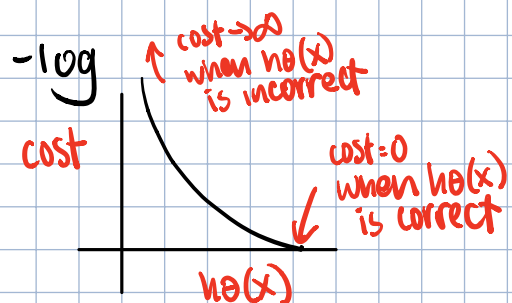


This is the
negative log function graph



Why are these shapes great?

- Its convex
- can have zero cost
- we can model cost to approach infinity



Logistic Regression Cost Function

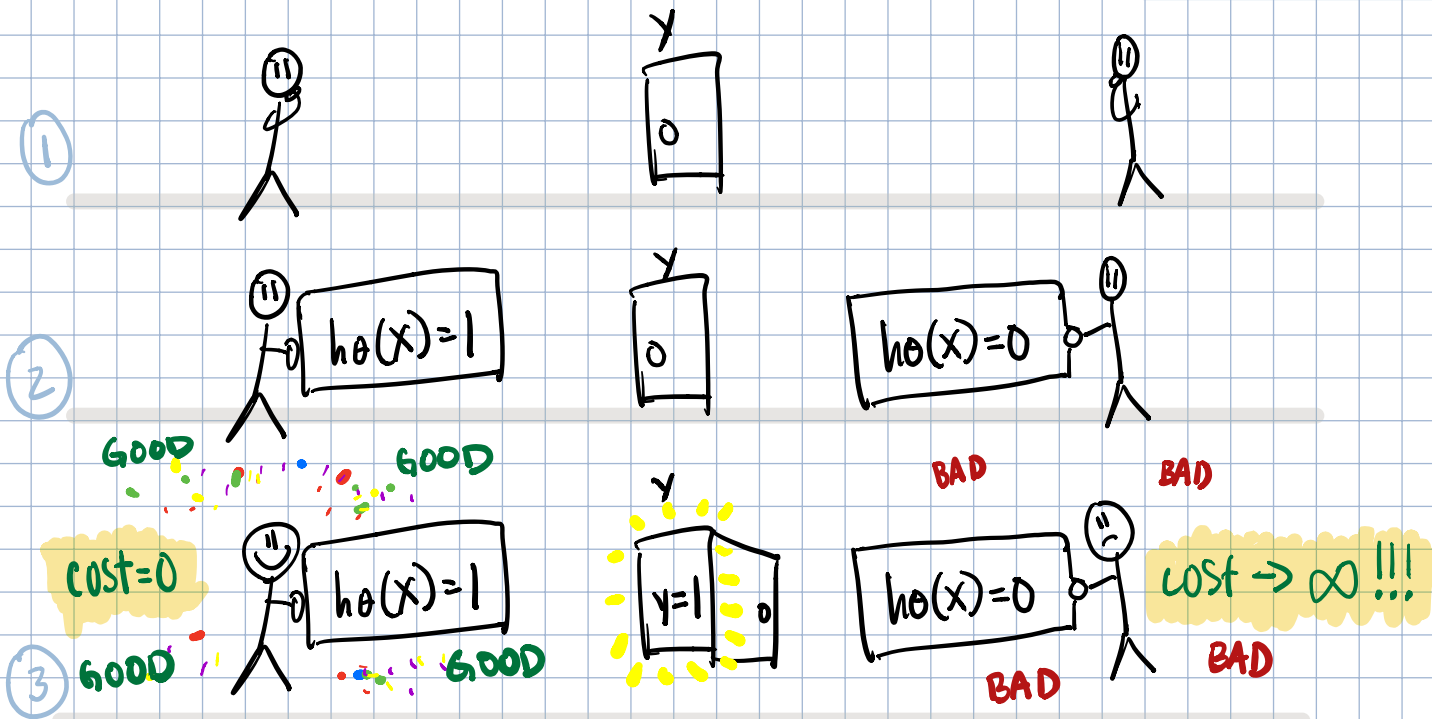
TLDR $\rightarrow \text{cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y=1 \\ -\log(1-h_{\theta}(x)) & \text{if } y=0 \end{cases}$

Intuition

if $y=1$, we want the correct prediction to "cost" less than the incorrect so...

$h_{\theta}(x)=1$ to "cost" less than $h_{\theta}(x)=0$

A TALE OF TWO HYPOTHESIS PREDICTIONS



In notation

cost = 0 if $y=1, h_{\theta}(x)=1$

but as $h_{\theta}(x) \rightarrow 0$

cost $\rightarrow \infty$

($h_{\theta}(x)$ approaches zero)

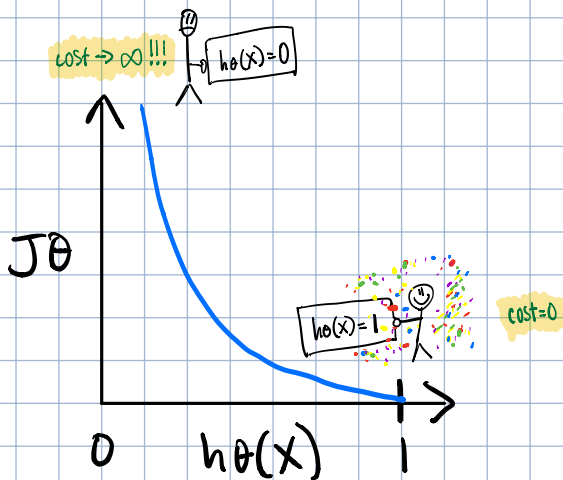
(cost approaches infinity)

visualizing the cost

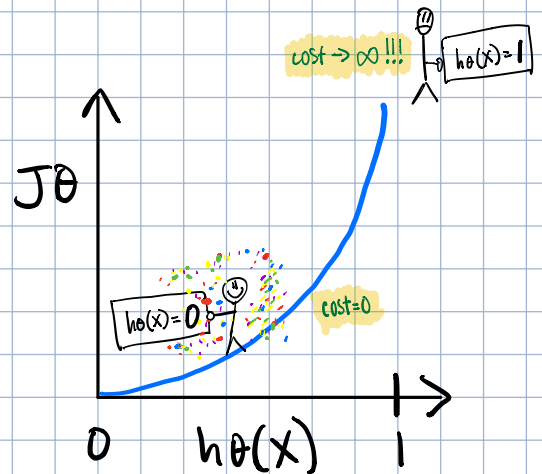
Logistic Regression Cost Function

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

When $y=1$ 
 $\text{cost} = -\log(h_{\theta}(x))$



When $y=0$ 
 $\text{cost} = -\log(1-h_{\theta}(x))$



★ 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 $J\theta$

Logistic Regression Cost Function

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

$$\text{cost}(h_{\theta}(x), y) = (y) \cdot (-\log(h_{\theta}(x))) + (1-y) \cdot (-\log(1-h_{\theta}(x)))$$

if $y=0$
this portion = 0

if $y=1$
this portion = 0

↙ rearrange (-) sign ↙ rearrange (-) sign

$$= -y \log(h_{\theta}(x)) - (1-y) \log(1-h_{\theta}(x))$$

extract (-) sign further (transitive prop)

$$= -1(y \log(h_{\theta}(x)) + (1-y) \log(1-h_{\theta}(x)))$$

one line!

We use this in $J\theta$ to
measure average cost

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

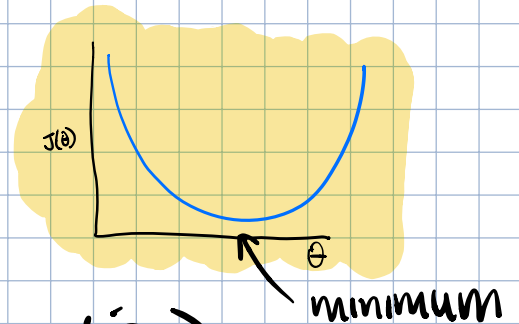
$$J\theta = \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1-y^{(i)}) \log(1-h_{\theta}(x^{(i)})) \right]$$

Minimizing Cost $J(\theta)$



Like always, given parameters θ
we want to minimize cost

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

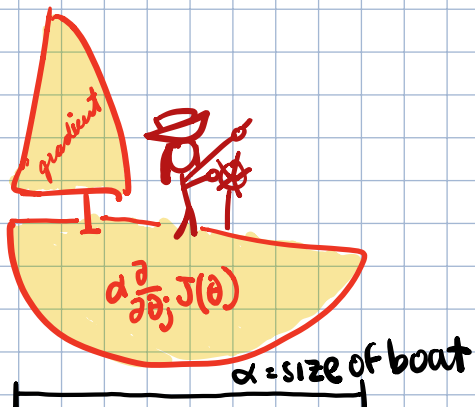


* we have a convex shape (function)
and we want to find the min value

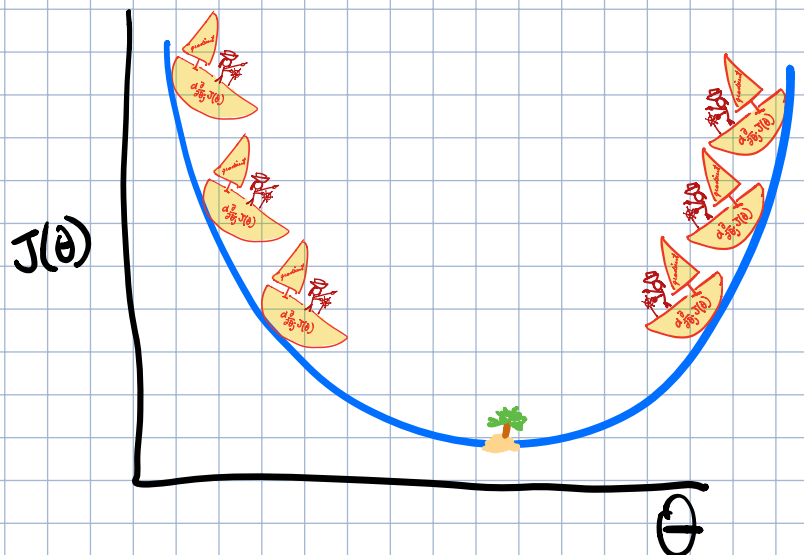
Welcome Back
Gradient Descent

$$\text{Repeat } \left\{ \begin{array}{l} \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \end{array} \right. \leftarrow \text{until convergence}$$

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

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

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

so...

Gradient Descent formula

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

(simultaneously update all θ_j)



repeat

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

← If gradient descent were a 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 algo's yourself
you can use these algo's without fully understanding the implementation

How to use 'fminunc' in Octave

① create a cost function

```
function [jVal, gradient] = costFunction(theta)
```

```
jVal = [... code to calc cost]
```

```
gradient = [... code to compute derivatives]
```

```
end
```

Ⓜ see gradient descent captain

② choose fminunc options

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
```

③ initialize a theta vector to zeros

```
initialTheta = zeros(2,1);
```

④ calculate

```
[optTheta, functionVal, exitFlag] =
```

```
fminunc(@costFunction, initialTheta, options)
```

[optTheta, ← optimum value of theta

functionVal, ← this should return close to 0

exitFlag] = ← 1 means this has convergence

fminunc(

@costFunction, ← pointer to function defined in step 1

initialTheta,

options)

Multi class Classification

what is it?

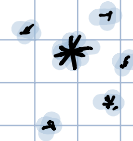
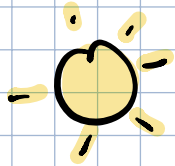
Examples

email : work, friends, family

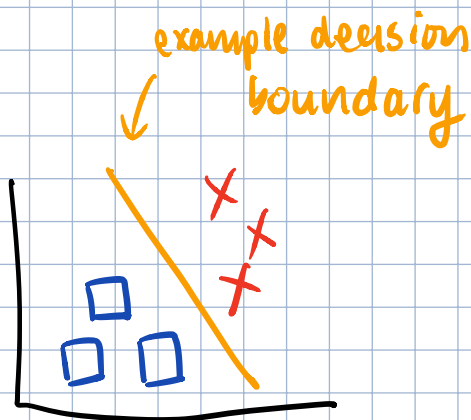
↑ ↑ ↑
 $y=1$ $y=2$ $y=3$

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

weather: sunny, cloudy, snowy

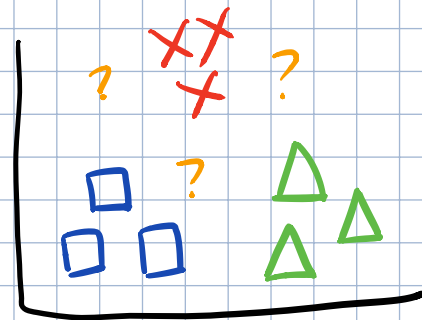


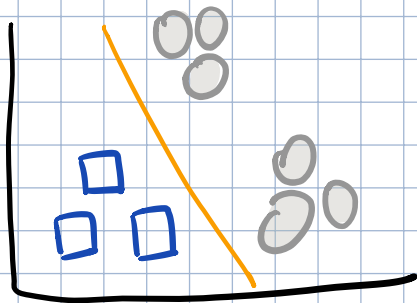
Instead of binary classification



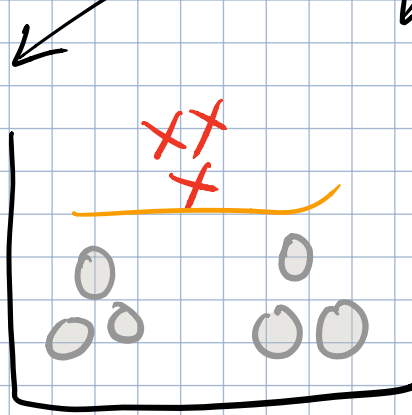
we have many classes

"One vs All" classification helps convert this multiclass problem into many binary class problems

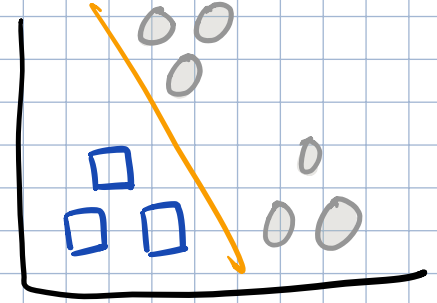




$$h_{\theta}^{(0)}(x) = P(y=0|x;\theta)$$



$$h_{\theta}^{(1)}(x) = P(y=1|x;\theta)$$



$$h_{\theta}^{(2)}(x) = P(y=2|x;\theta)$$

Q: How?

A: given $n+1$ classes $\dots \dots \dots y \in \{0, 1, \dots, n\}$

we solve a hypothesis

function for each class $\dots \dots h_{\theta}^{(0)}(x) = P(y=0|x;\theta)$

$$h_{\theta}^{(1)}(x) = P(y=1|x;\theta)$$

$$h_{\theta}^{(2)}(x) = P(y=2|x;\theta)$$

...

$$h_{\theta}^{(n)}(x) = P(y=n|x;\theta)$$

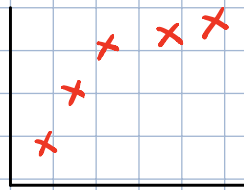
🤔 given any new input example we can run it through all of our hypothesis functions and choose the y (class) with the highest probability

OVERfitting

Q: What is overfitting?

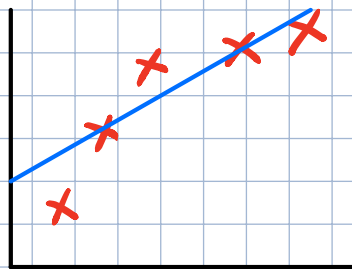
A: If we have too many features, the learned hypothesis may fit the training data so well that it fails to "generalize" to new example inputs

Given this data
we can fit a linear,
quadratic, even a
higher order function



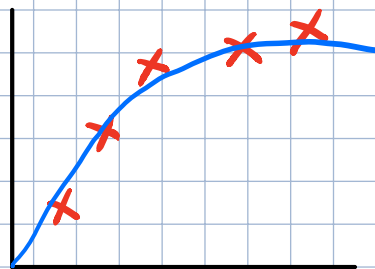
linear
regression

$$\theta_0 + \theta_1 x$$



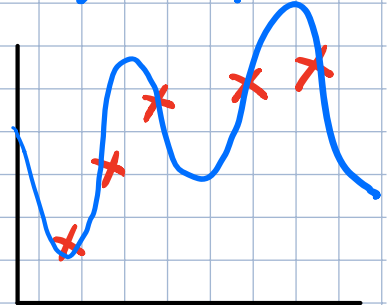
↑
underfit
"high bias"

$$\theta_0 + \theta_1 x + \theta_2 x^2$$



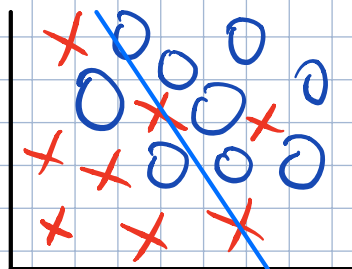
↑
"JUST RIGHT"

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

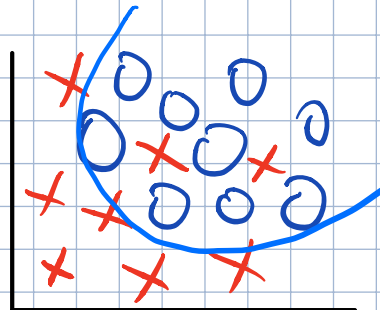


↑
overfit
"high variance"

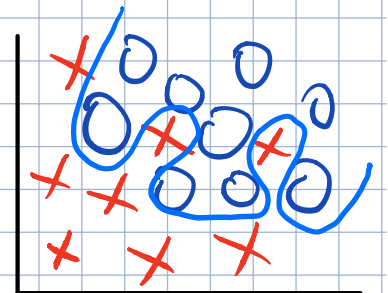
logistic
regression



↑
underfit
"high bias"



↑
"JUST RIGHT"



↑
overfit
"high variance"

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

① manually remove features

② choose a feature selection algo

② regularization

↑ TODO

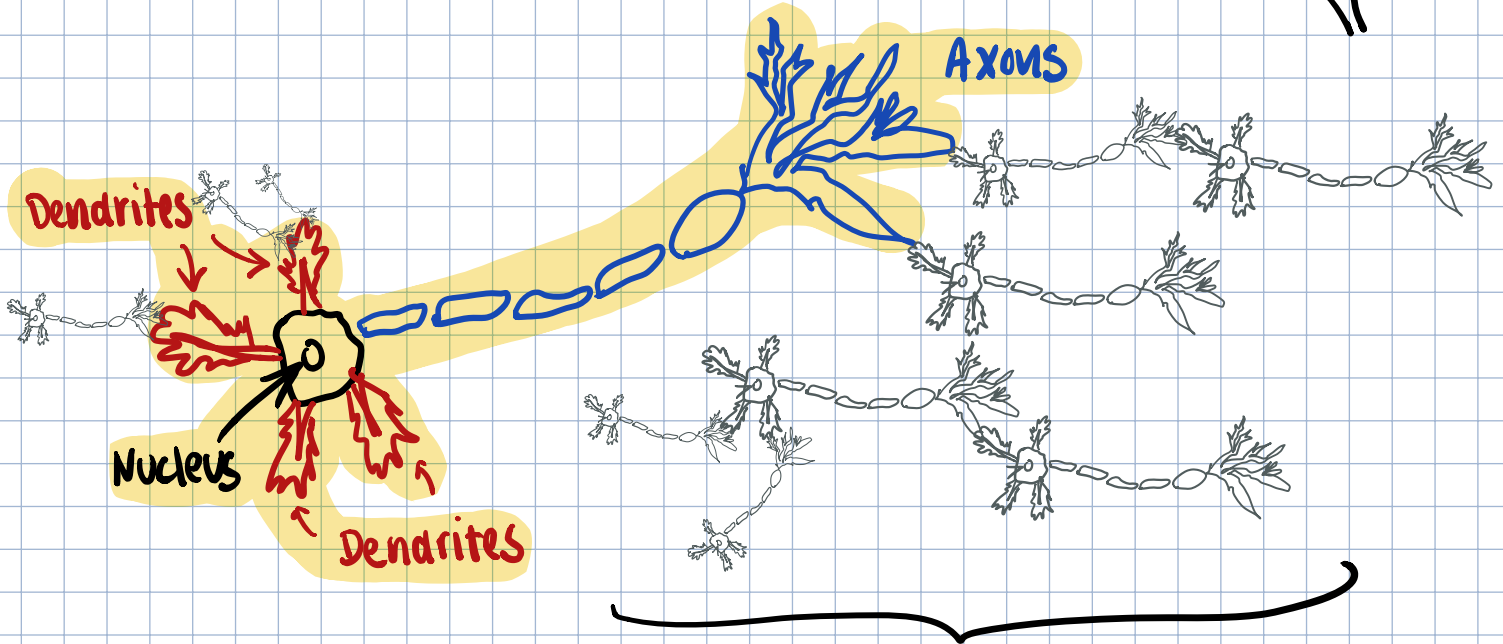
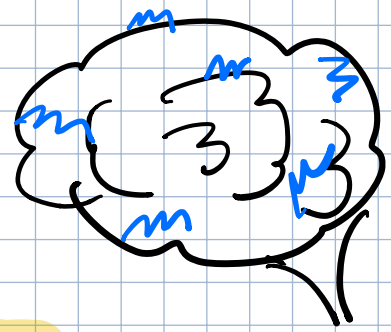
week 4

Neurons in the brain

The neurons that make up our brain have

Dendrites - receivers of input

Axons - broadcasters of output

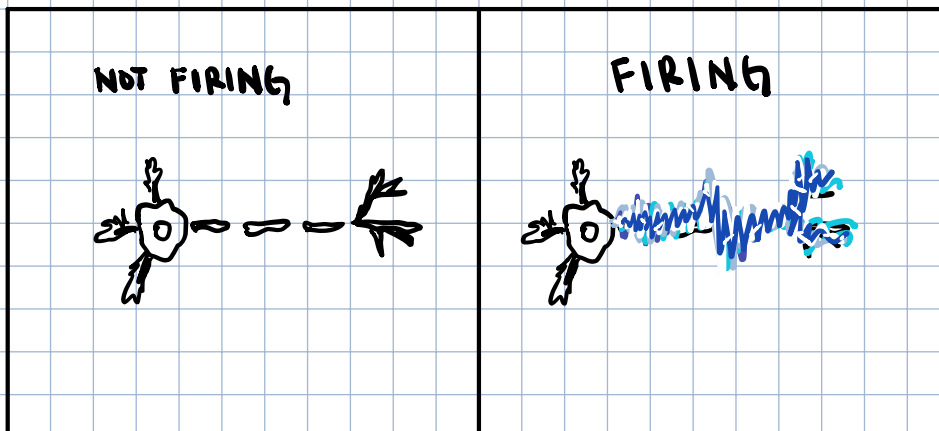


Note that each neuron can output to many other neurons and serve as inputs to those receiving neurons.

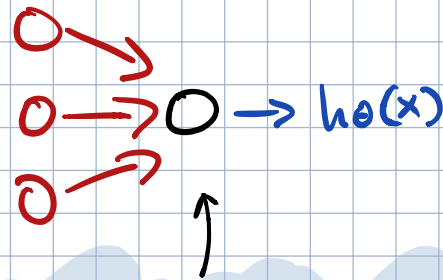
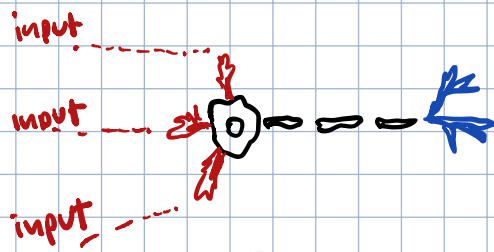
Q: How do neurons communicate?

A: An electric pulse is "fired" from one neuron to the other.

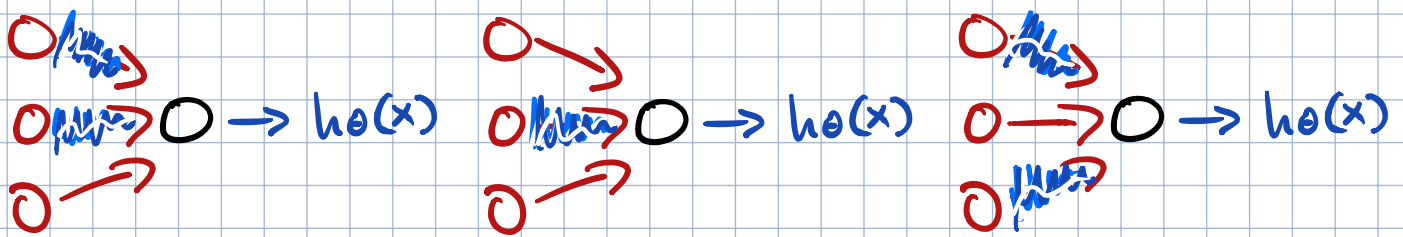
Note: neurons fire or they don't. There is no partial firing!



a simple model based $\Rightarrow$... can be simplified to:
on this neuron



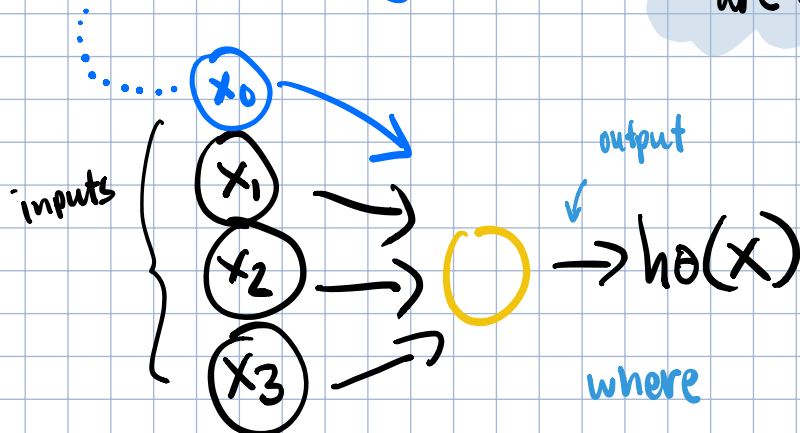
This is a neuron or an "artificial neuron" with a sigmoid or logistic activation function



Single neuron model: logistic unit

sometimes x_0 is drawn but since this bias unit is always 1 this isn't necessary

There is a theta value for every input $x^{(i)}$, in neural networks these parameters are also referred to as "weights"



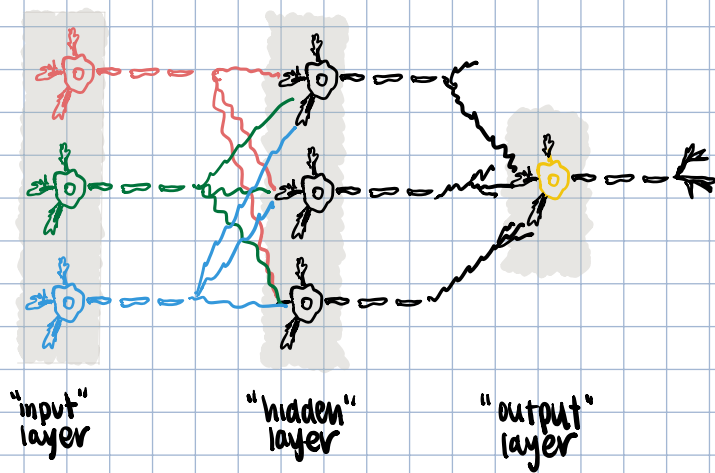
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

"weights"

where

$$h_0(x) = \frac{1}{1 + e^{-\theta^T x}}$$

we can now expand our model to many outputs into "layers" of neurons

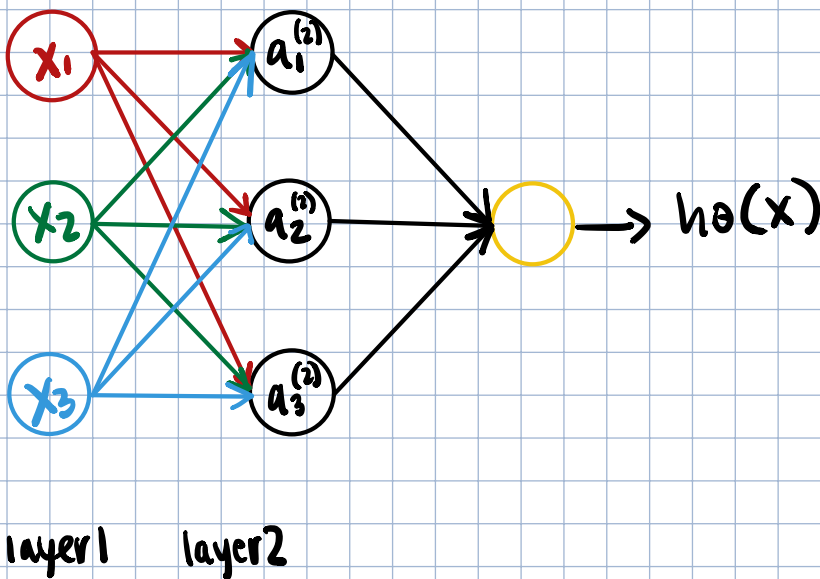


terminology

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

The last layer is the **output layer**

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



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

Note: theta is now stored in a matrix $\theta = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$

In the single neuron example (like logistic regression) we only required a vector of theta values.

The layers of an NN calculate to many nodes and require a matrix of theta values

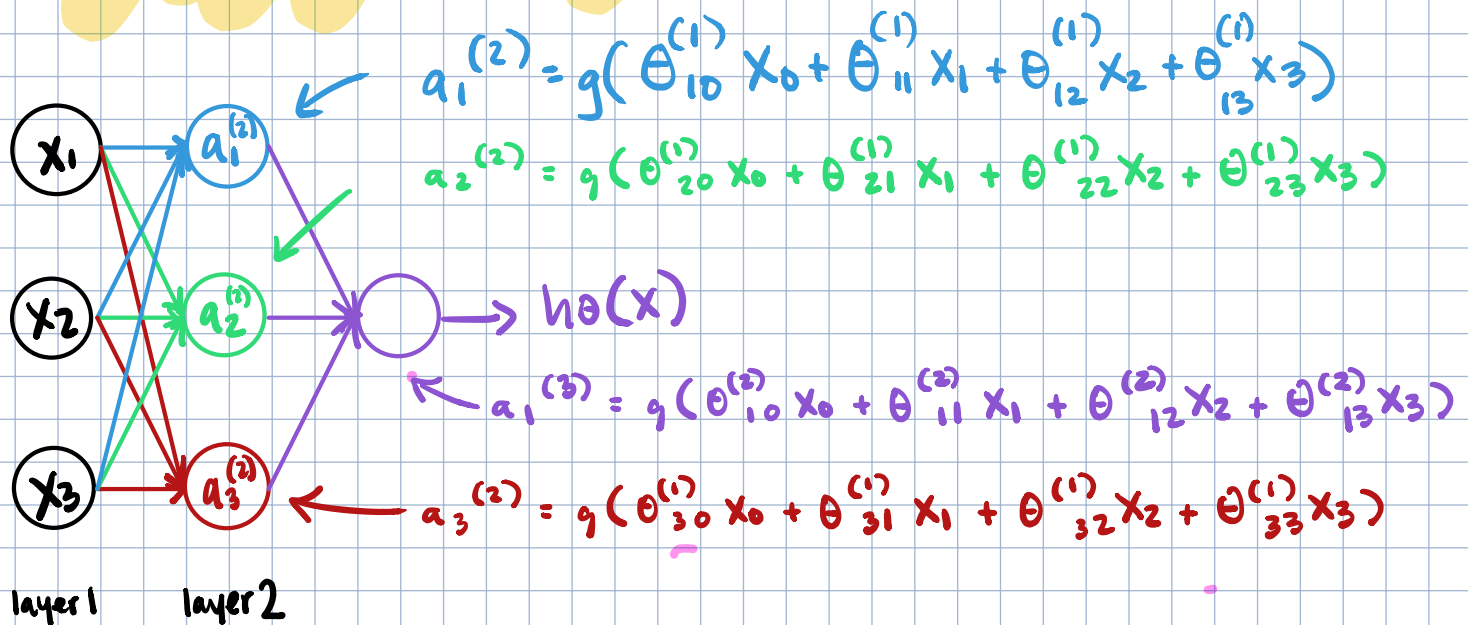
Notation

This is a single theta value

Θ layer to node from node

theta from layer 1 to 2

$\Theta_{30}^{(1)}$ for the 0th node of the layer (1) for the 3rd node of the next layer (2)



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

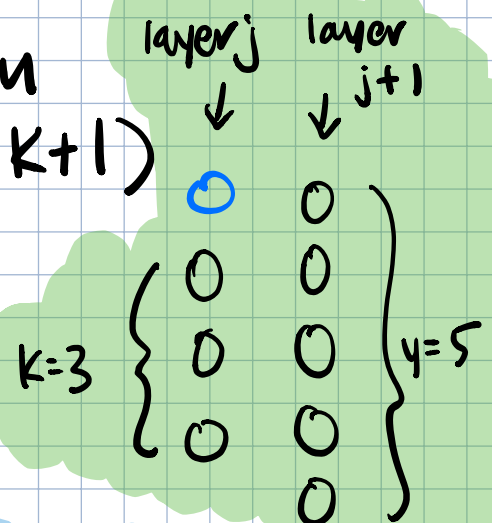
A: If a network has K units layer j and y units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $(y \times K+1)$

Example:

if layer 1 has 3 units and layer 2 has 5 units

$\Theta^{(1)}$ will have dimensions (5×4)

dimensions are: (units in next layer $\times$ units in current layer plus bias unit)



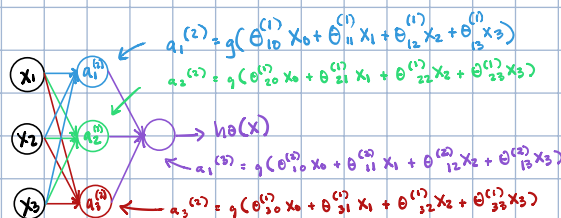
$$\Theta = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{bmatrix}$$

Lecture 2

Q: How do we calculate the layers efficiently?

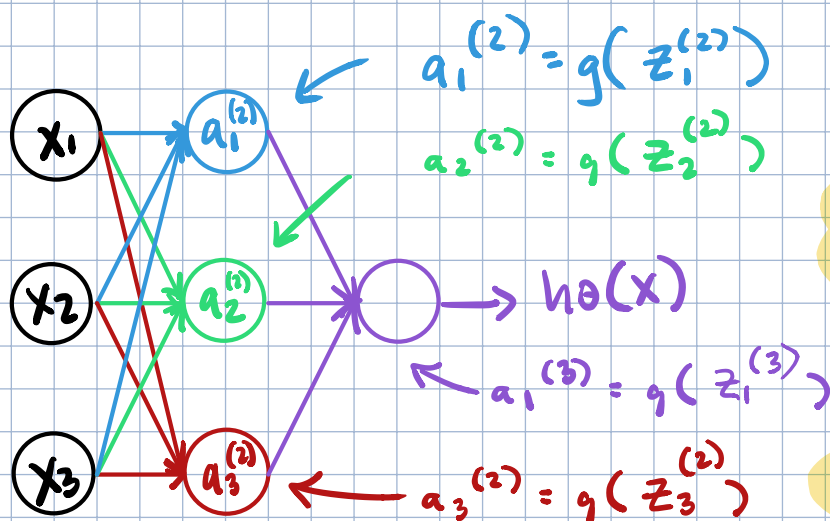
A: vectorized Implementation of Forward Propagation
 "Forward Propagation": vectorized Implementation

Remember this?



lets convert this into vectors

As a simplification, z now represents the $\theta \cdot X$ linear combination



Notation

single linear combination

$z_2^{(2)}$ ← layer 2
 linear combination of the 2nd node

vector of linear combinations of layer

$z^{(3)}$ ← layer 3

Summary

nodes of layer 2

$$\begin{cases} a_1^{(2)} = g(z_1^{(2)}) \\ a_2^{(2)} = g(z_2^{(2)}) \\ a_3^{(2)} = g(z_3^{(2)}) \end{cases}$$

This can be represented by vector $V^{(2)}$
 where $g(z)$ is an element wise sigmoid operation

Say we have input vector x

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

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Goal: find vector $z^{(2)}$ and $a^{(2)}$

$$z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

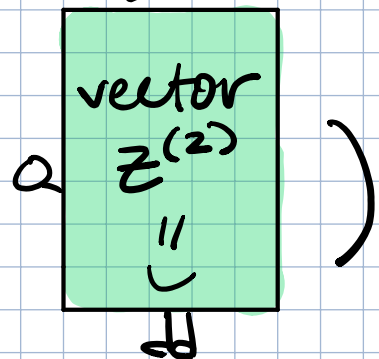
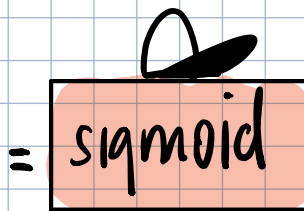
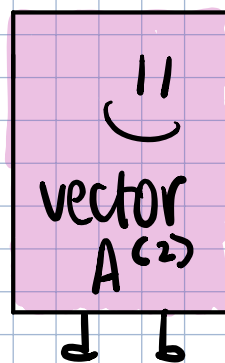
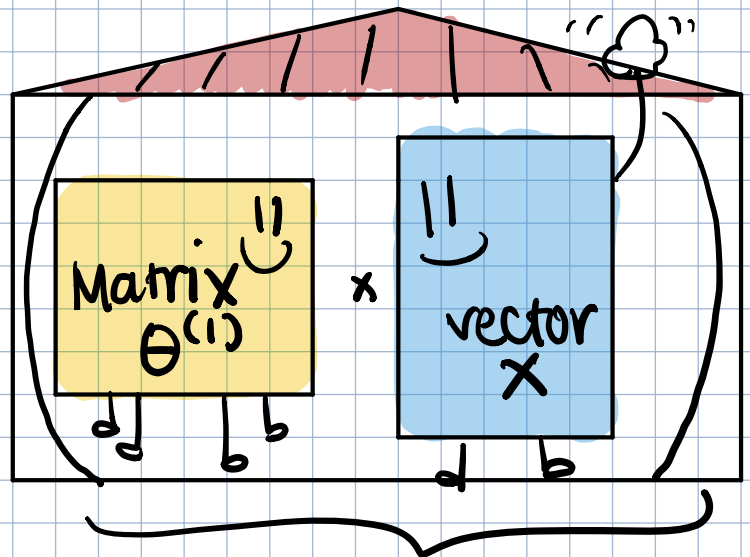
$$a^{(2)} = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

Solution:

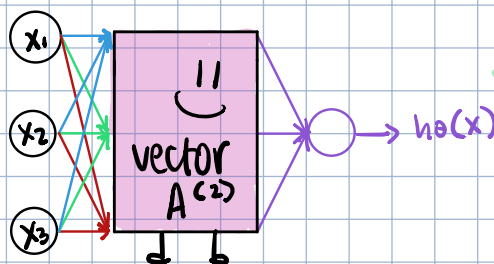
perform a weighted linear combination

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

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



The input layer can also be called a "layer 1" or "activation layer 1"



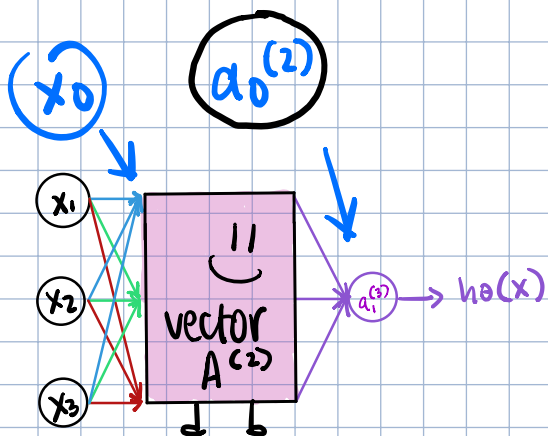
This process "feeds forward" into all further layers



HALT!

We forgot to mention... the Bias Unit

The bias unit is always = 1 so we don't have to calculate it directly, but we have to include it in calculations



Q: How do we calculate the output layer $a^{(3)}$?

A: Same as all other layers

step 1: add a bias unit ($a_0^{(2)} = 1$)

step 2: weighted linear combination

$$z^{(3)} = \theta^{(2)} a^{(2)}$$

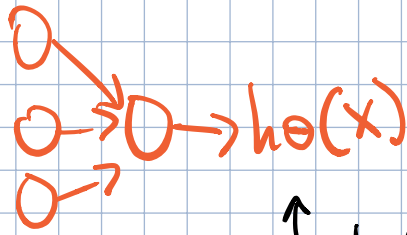
step 3: element wise sigmoid function

$$a^{(3)} = g(z^{(3)})$$

$$h_0(x) = a^{(3)}$$

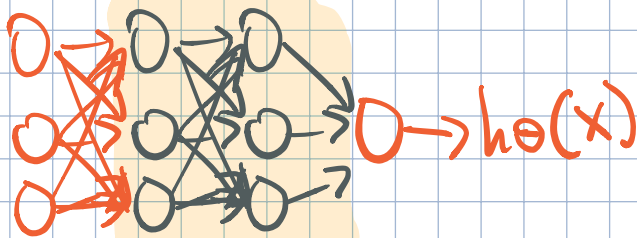
≡ The Neural Network is learning its own features! ≡

Think back to logistic regression

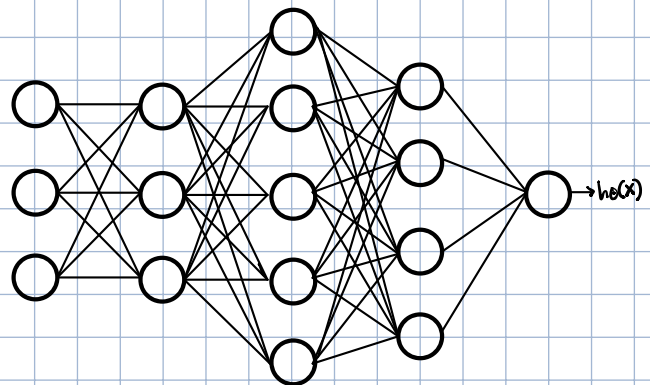
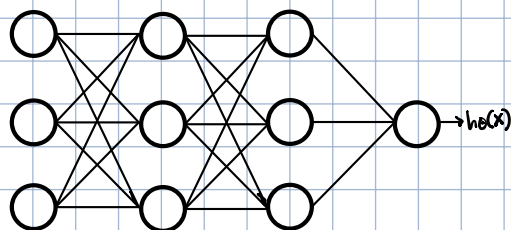


↑ $h_{\theta}(x)$ could only operate on the given inputs

As we add hidden layers, our neural network is generating new inputs/features for the next layer and eventually generating its own inputs for $h_{\theta}(x)$



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



☆ this allows for more complex hypothesis outputs than the raw features alone a simple higher order polynomial $h_{\theta}(x)$

Neural Network: Examples and Intuitions

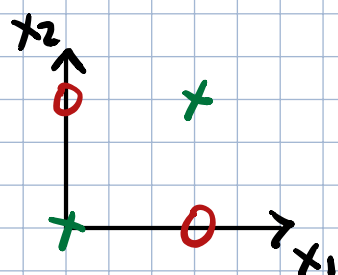
NN's allow us to model complex relationships that cannot be easily modeled as linear combinations

Two complex logical functions are

① XOR - exclusive OR

② XNOR - the inverse of XOR

What is XOR (exclusive OR)?



XOR returns true

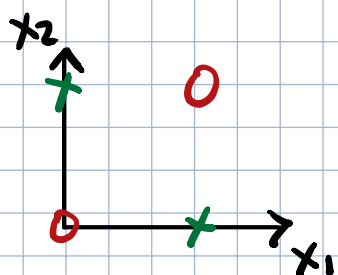
if x_1 OR x_2 are true
but not if

x_1 AND x_2 are true

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

What is XNOR?

note the inverse

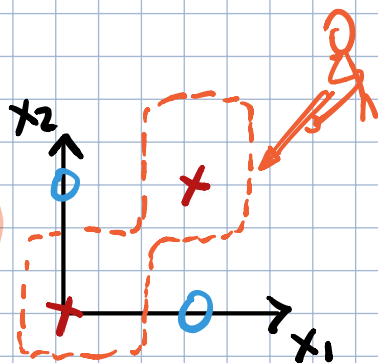


XNOR returns true

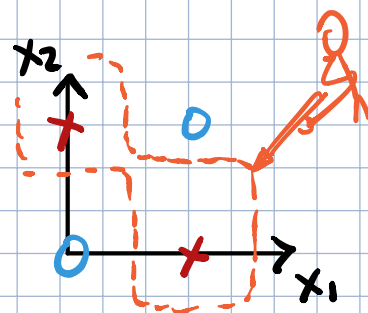
if x_1 AND x_2 are true
or if x_1 AND x_2 are false

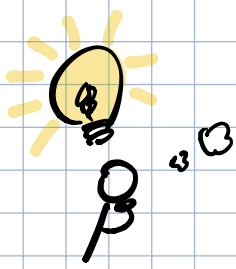
x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1

Imagine the difficulty in drawing a decision boundary



XNOR
XOR





We can use simpler logical functions to build XOR

AND

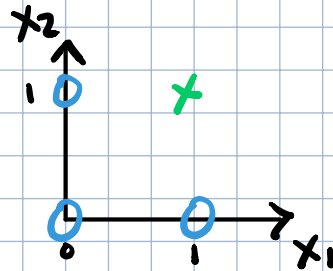
OR

NOT

AND OR NOT as building blocks

AND

use: $y = x_1 \text{ AND } x_2$
 y is true if both
 x_1 and x_2 are true

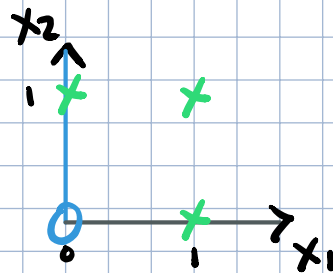


Truth Table

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

OR

use: $y = x_1 \text{ OR } x_2$
 y is true if either
 x_1 or x_2 are true



Truth Table

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

NOT

use: $y = \text{NOT } x$
 y is true if x is false
 y is false if x is true

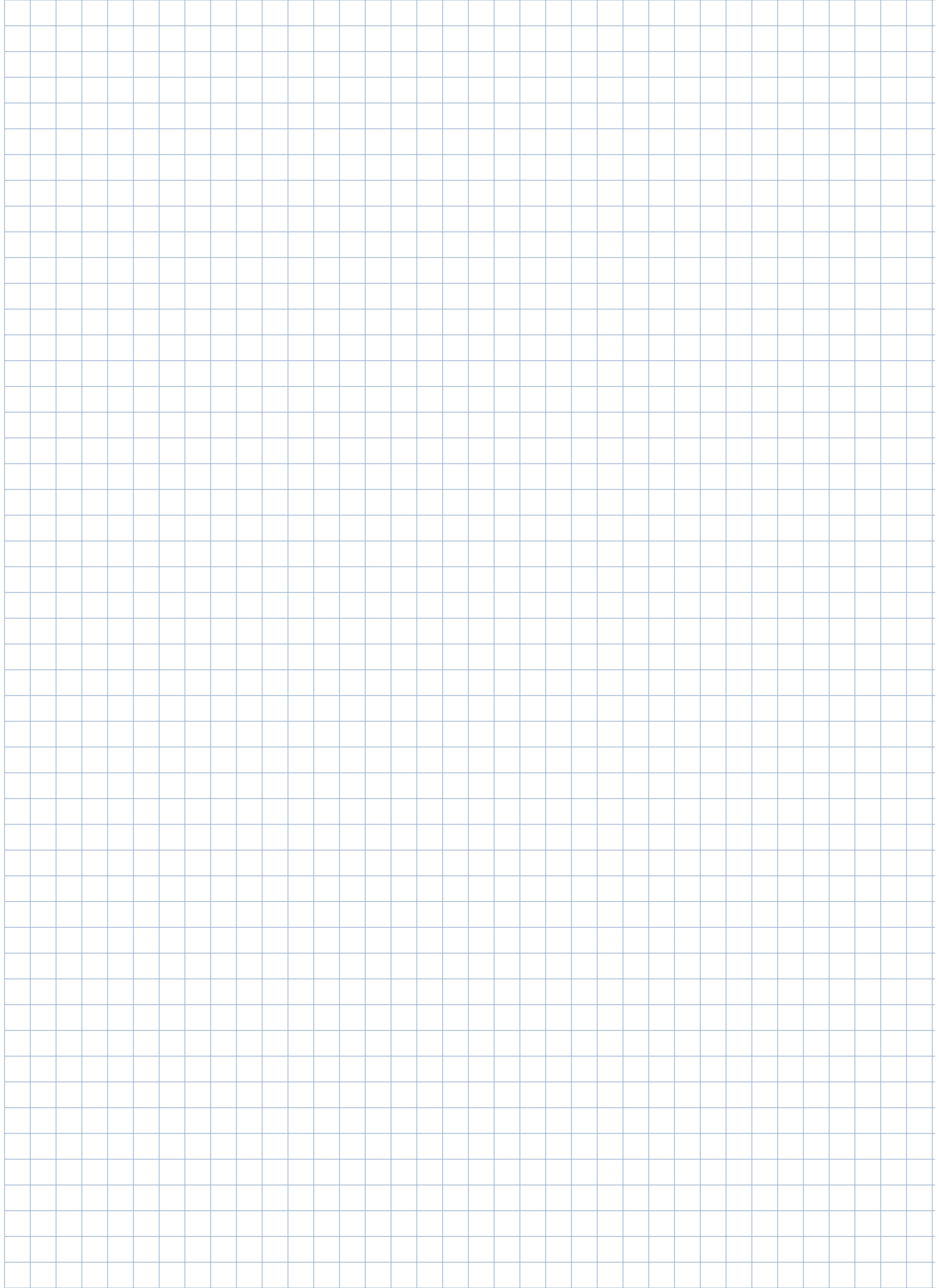


Truth Table

x	y
0	1
1	0

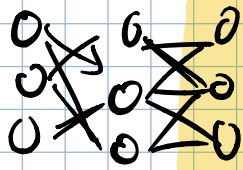
"Building" XOR with AND, OR and NOT

$$\text{XOR} = (x_1 \text{ OR } x_2) \text{ AND } (\text{NOT } (x_1 \text{ AND } x_2))$$

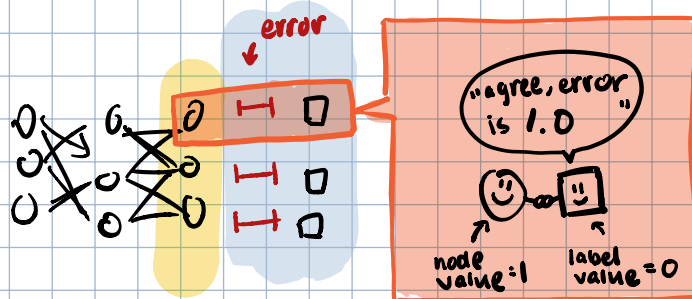


week 5

Introducing backpropagation and why
through the feed forward mechanism
eventually our NN creates an output layer



This output can be directly compared to
the actual label values from the training set.

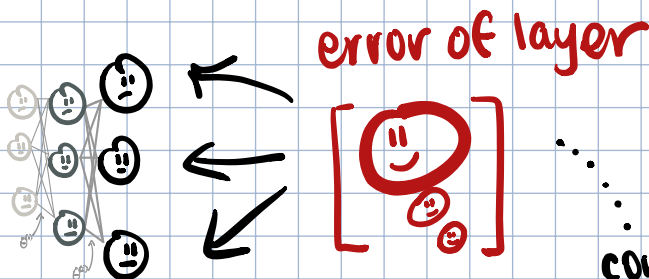


Vectorized Formula

activation nodes of output layer - training data y values = error of output layer

error of layer
[😊]

Q: Now that we know the error,
how can we use this info
to update our neural network?



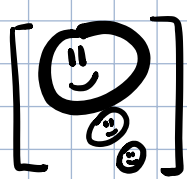
consider...
activations and weights
feed forward by layer
errors propagate backwards

This input layer does
not generate errors

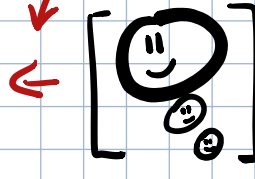
in backprop because we don't change this

by layer

"we are perfect!"



error of layer



error of layer



Q: How much of the error do we backprop to each node?



Insight: the layers are built by feed forward as a function of theta and activation values

$$\begin{bmatrix} \text{smiley face} \\ \text{smiley face} \\ \text{smiley face} \end{bmatrix} g(\theta^{(n)} \cdot a^{(n)}) \Rightarrow \begin{bmatrix} \text{smiley face} \\ \text{smiley face} \\ \text{smiley face} \end{bmatrix}$$

how much to propagate can be a function of the theta values and the activation values

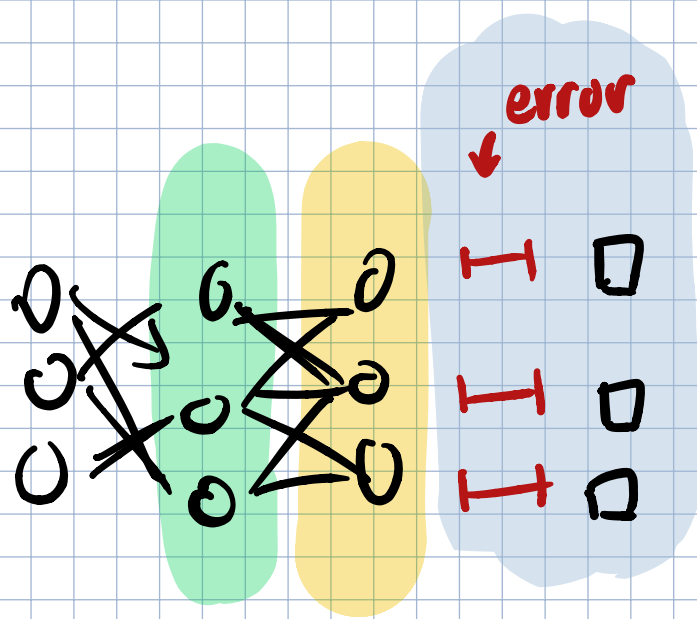
Calculating error for layers

output layer error = output layer - labels

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

hidden layer error is a function of θ and values

$$\delta^{(n)} = (\theta^{(n)})^T \delta^{(n+1)} \cdot * g'(z^{(n)})$$



$$\begin{array}{c}
 a^{(1)} \quad a^{(2)} \quad a^{(3)} \\
 \vdots \quad \vdots \quad \vdots \\
 \delta^{(3)} = a^{(3)} - y \\
 \vdots \\
 \delta^{(2)} = (\theta^{(2)})^T \delta^{(3)} \cdot * \quad g'(z^{(2)}) \\
 \uparrow \quad \uparrow \quad \uparrow \quad \uparrow \\
 \text{theta} \quad \text{error} \quad \text{dot} \quad \text{activation} \\
 \text{used to} \quad \text{of next} \quad \text{product} \quad \text{layer} \\
 \text{compute} \quad \text{layer} \quad \quad \quad \text{values} \\
 \text{next layer} \\
 \text{remember: } a^{(2)} = g(z^{(2)}) \\
 \text{input layer error is not calculated}
 \end{array}$$

If this is difficult, you are not alone

"I've actually used backpropagation 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 backpropagation is doing"

- Andrew Ng