machine Learning   Prof. Anderson Rocha.

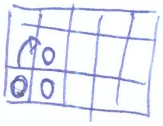
1. Grew out of work in AI
2. New capabilities for computers

Some examples

1. Database mining, large datasets, web automation, web click data, medical records, social networks
2. Applications can't program by hand
   - autonomous car, helicopter
   - handwritten recognition
   - NLP
   - vision
3. self-customizing
   - Amazon
   - Netflix product recommendations
4. understanding human learning.

What is ML

1. field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel)



checkers.

2. Tom mitchell : well-posed learning program ⇒ a computer program is said to learn from exp Ⓔ wrt task Ⓣ and some perf. measure Ⓟ if its perf. on Ⓣ as measured by Ⓟ, improves with exp. Ⓔ.

Types of ML

- SL vs SSL vs unsup. learning
- Reinf learning, recommender systems

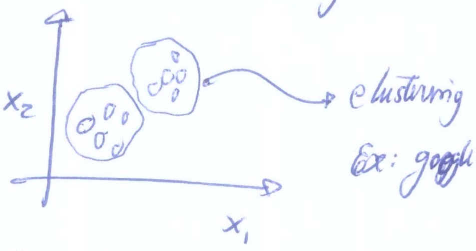
Practical advice : how to use and where to apply ML.

Supervised Learning

- Regression ⇒ predict cont. output.
- classif ⇒ predict discrete value (class) output.

unsupervised Learning

find some structure on the data



Ex: google news.

Example: - Genes x Individuals
- datacenters.
- cohesive groups of people on social networks
- Group customers
- Astronomical data analysis (how galaxies are formed).

Cocktail party problem ⇒ overlapping voices.

Languages
- Octave
- matlab
- R

} first prototyping then ⇝ other languages.

Notation

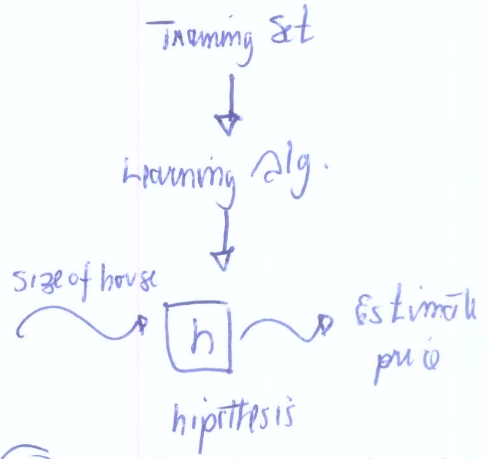
(m): nbr of training examples

x's input variables/features

y's out. variable/target variable

(x,y) one training example.

$(x^{(i)}, y^{(i)})$ ith training example.

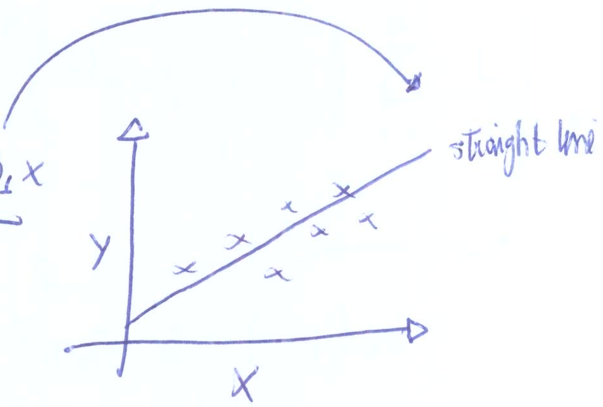


(h) maps from x's to y's.

How to represent $\boxed{h}$?

Initial choice $h_\theta(x) = \underline{\theta_0 + \theta_1 x}$

shorthand $h(X)$



why linear function => simple building block.

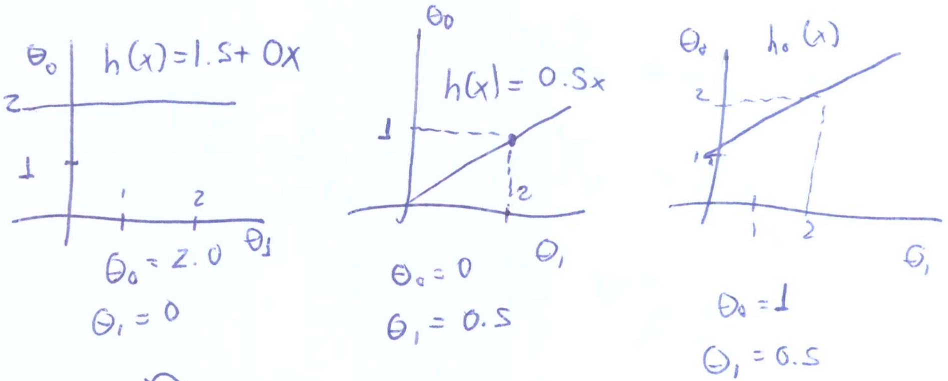
Here we have an example of a linear regression with one variable or

$\underbrace{\text{univariate}}_{\text{1 variable}}$ linear regression.

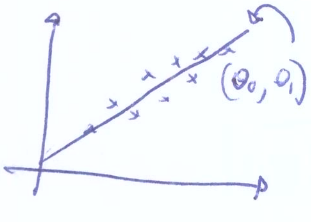
Defining a cost function

hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$ θ_s parameters

3 examples of hypothesis :



In linear Regression



$h_\theta(x)$ must be $\underline{\underline{\text{close}}}$ to the training data.

More formally

minimize θ_0, θ_1 $(h_\theta(x) - y)^2$.

$$\min_{\theta_0 \theta_1} \bigcirc \sum_{i=1}^{\boxed{M}} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2.$$

$\frac{1}{2M}$ only for make it easier.

Examples in the traing.

$$\underbrace{\min_{\theta_0 \theta_1} \left(\frac{1}{2M} \sum_{i=1}^{M} \left(h_\theta (x^{(i)}) - y^{(i)} \right)^2 \right)}_{} = J(\theta_0, \theta_1).$$

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

with this, we define a cost function $J(\theta_0, \theta_1)$.

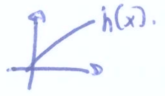
so we can simply say $\boxed{\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)}$ — cost function called squared Error cost function. most used for Regression.

SSD.

Cost function intuition

① hypothesis $h_\theta(x) = \theta_0 + \theta_1 x$.

② Params θ_0, θ_1

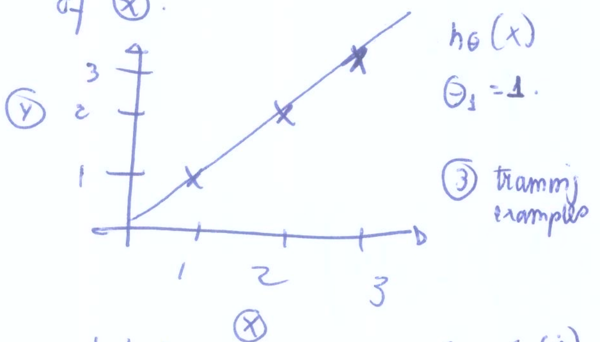


③ Cost function $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta (x^{(i)}) - y^{(i)})^2$

④ Goal: $\underset{\theta_0, \theta_1}{\text{minimize}} \; J(\theta_0, \theta_1)$

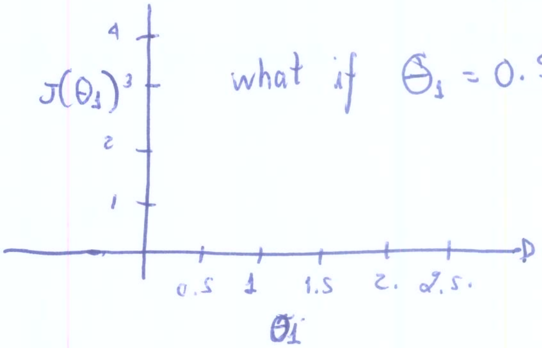
If we simplify $h_\theta(x) = \theta_1 x$. $\theta_0 = 0$.

$\boxed{h_\theta(x)}$ for fixed θ_1, this is a function of x.

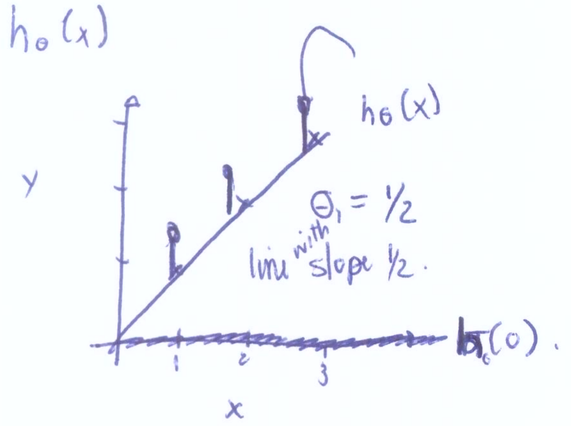


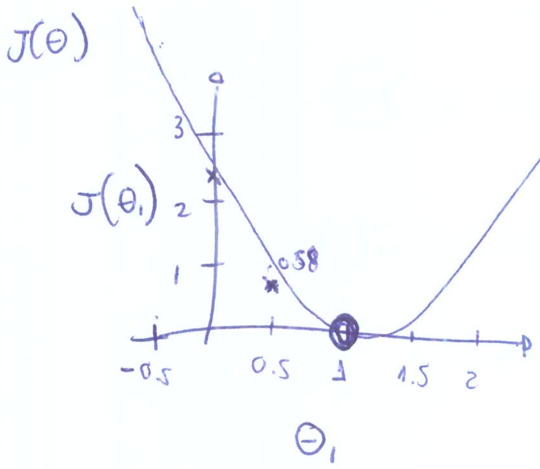
what is $J(\theta_1)$? $\frac{1}{2m} (\theta_1 (x^{(i)} - y^{(i)})^2$
$= (0^2 + 0^2 + 0^2) \frac{1}{2m} = \boxed{0}$

$\boxed{J(\theta_1)}$ is a function of the param θ_1.



what if $\theta_1 = 0.5$?





$$J(0.5) = \frac{1}{2M}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$$

$$= \frac{1}{2\times 3}(3.5) \cong \boxed{0.58}.$$

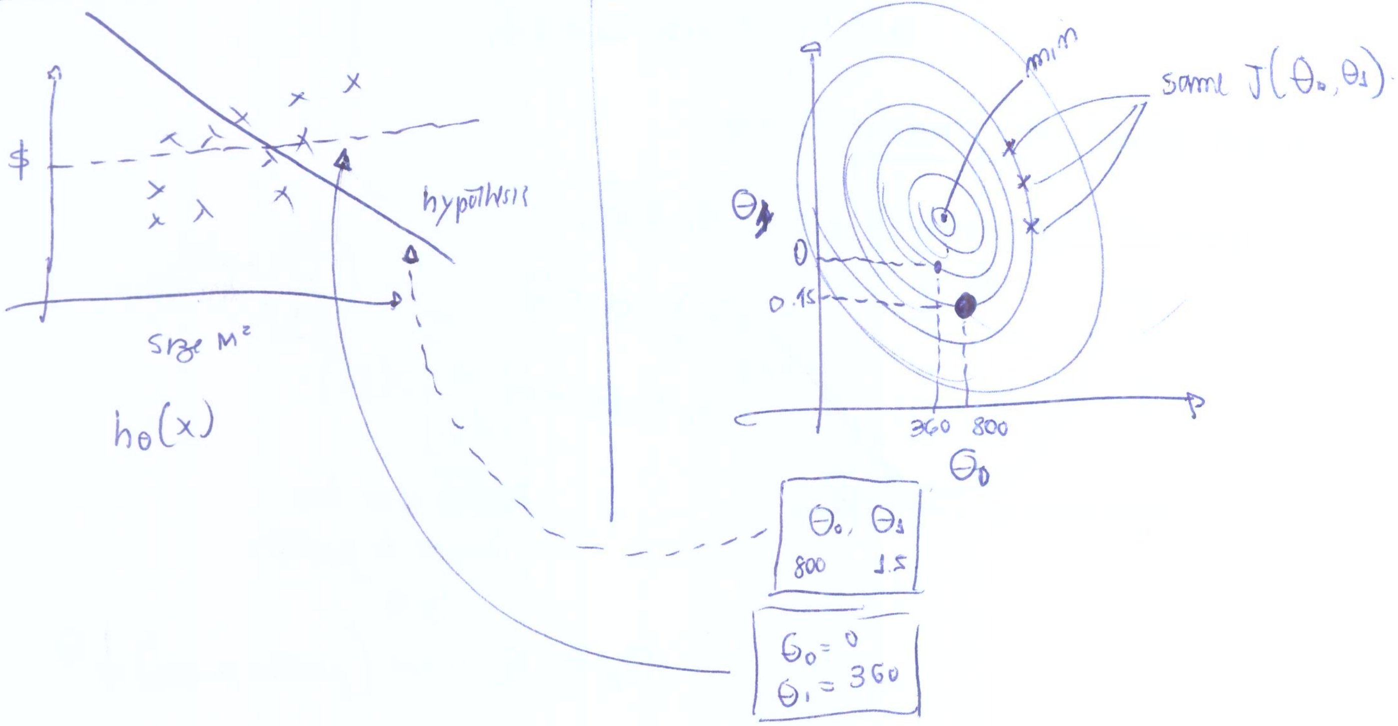
$J(0) = ?$

$$J(0) = \frac{1}{2m}\left(1^2 + 2^2 + 3^2\right)$$

$$\frac{1}{2m} \cdot 14 = \boxed{2.3}.$$

for each value of θ_1, we have a different hypothesis for $h_\theta(x)$ (different line).

cost function (contour plots)



Gradient Descent

Used everywhere in ML

1. Have some function $J(\theta_0, \theta_1)$

2. $\min\limits_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

3. Outline
   (a) start with some θ_0, θ_1
   (b) Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum.

$\rightsquigarrow$ what we need to define is α (the learning rate).

Implementation

Correct form

$$t_0 \leftarrow \theta_0 - \overbrace{\alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)}^{\text{partial derivatives}}$$

$$t_1 \leftarrow \theta_1 - \alpha \cdot \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\left.\begin{array}{l} \theta_0 \leftarrow t_0 \\ \theta_1 \leftarrow t_1 \end{array}\right\} \text{simultaneously update both } \theta_0 \text{ and } \theta_1$$

Incorrect form

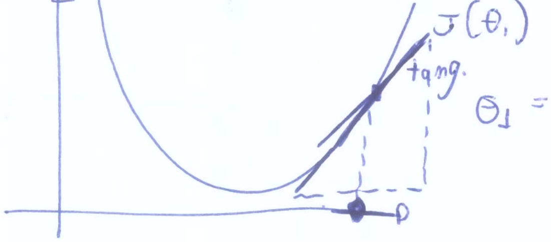
~~temp~~ $t_0 \leftarrow \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$

$\theta_0 \leftarrow t_0$

$\vdots$

Intuition about the Gradient Descent

Suppose we have $\min\limits_{\theta_1} J(\theta_1)$ $\theta_1 \in \mathbb{R}$.



$\theta_1 \in \mathbb{R}$.

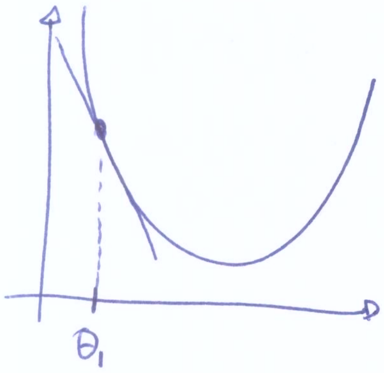
$\boxed{\frac{\partial}{\partial \theta}}$ partial derivative.

$$\theta_1 = \theta_1 - \alpha \underbrace{\overbrace{\boxed{\frac{d}{d\theta_1}}}^{\text{derivative}}}_{\text{slope}} J(\theta_1).$$

~~Tangent~~ of the line tangent to function.

≥ 0.

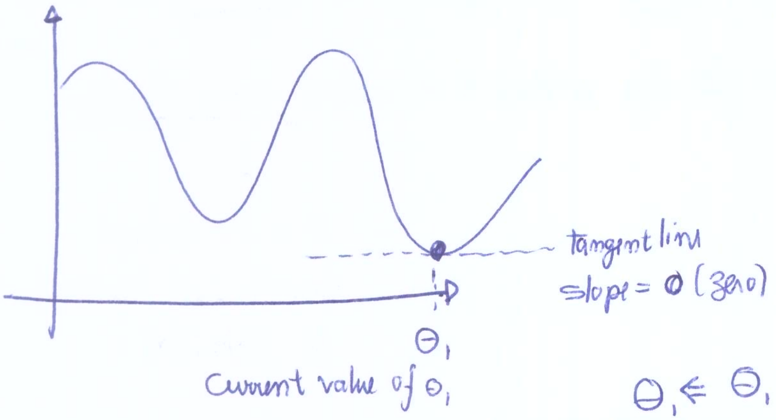
$\theta_1 \leftarrow \theta_1 - \alpha \cdot (\text{positive number}) \downarrow \theta$.



$$\frac{\partial J(\theta_1)}{d\theta_1} \le 0.$$

$$\theta_1 \leftarrow \theta_1 - \alpha(\text{neg. number}) \uparrow \theta.$$

→ What if GD is already at a minimum ?



$$\theta_1 \Leftarrow \theta_1 - \alpha \underbrace{\boxed{\frac{d}{d\theta_1} J(\theta_1)}}_{\text{zero}}$$

$$\theta_1 \leftarrow \theta_1 - \alpha . 0$$

$$\theta_1 \leftarrow \theta_1 .$$

$\boxed{\theta_1 \text{ stays unchanged}}$ that's why GD can converge to a local min even with α fixed.

GD automatically takes smaller steps as we approach a local min due to the derivative term. So, no need for decreasing α over time.

GD for linear regression

how to derive $\frac{\partial}{\partial\theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial\theta_j} \cdot \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 .$

$$= \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 \cdot x^{(i)} - y^{(i)} \right)^2.$$

$$\frac{\partial}{x} \left(x^2 - 2 \right)^2$$

$$2\left(x^2 - 2 \right) \cdot 2x$$

$$4x \left(x^2 - 2 \right).$$

$$\theta_0 \Rightarrow j = 0 : \frac{1}{m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right).$$

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

⇝ The cost function for ~~[illegible]~~ linear regression is always a bow shape function

⇝ The technical term for this is that it is called convex function

↓

- is a bow shape
- doesn't have local opt except for the global one.

The algorithm we just defined is what we call Batch Gradient Descent.

Batch : each step of GD uses all training examples.

⇝ there other version that use only ~~a few~~ subset of the training.

⇝ for the case of linear regression, there is a closed form solution (normal equations) but Gradient descent will scale better for larger datasets than normal equations.

⇝ The derivative is just the slope of the cost function J.

Generalization of Gradient Descent

Recalling, the GD algorithm is given by

repeat until convergence {

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

$$\theta_1 \leftarrow \theta_1 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta (x^{(i)}) - y^{(i)} \right) - x^{(i)}}_{\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1).}$$

}

→ It can be susceptible to local optima.

→ But the cost function for the linear regression is always convex (bow-shape)

→ only the global optimum.

There are two extensions of GD that we must consider

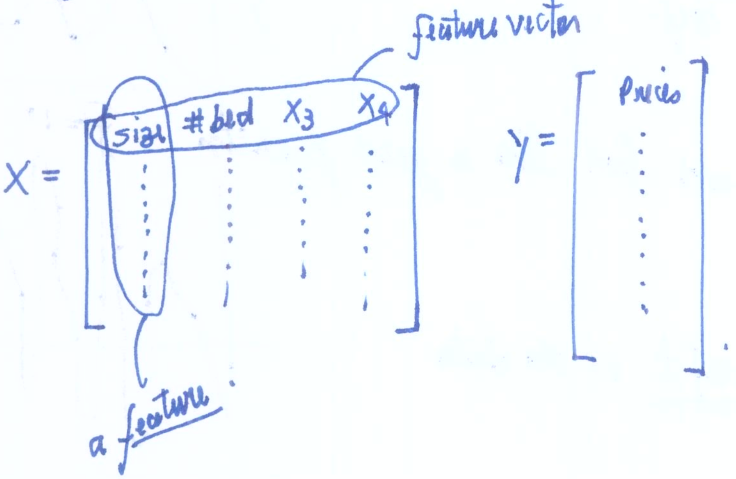
① In $\min J(\theta_0, \theta_1)$, solve for θ_0, θ_1 exactly, without iterative GD algorithm One advantage is that α is not necessary anymore but in some cases (~~large~~ high dim) the GD may be more appropriate.

② Longer ~~[illegible]~~ number of features. For instance

size, # bedrooms, # floors, # age of home ⇝ y (price).

x_1 x_2 x_3 x_4

Notation



Increasing the number of features

We have considered so far linear functions of several observed ~~variables~~ features

$x_1, x_2, \dots$

Suppose now that we have only one feature $\boxed{x_1}$ but we would like our predictor to be a non linear function of $\boxed{x}$ $\hat{y}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 \dots$

We can simply define new features $x_2 = x^2$, $x_3 = x^3$ just like we did with $x_0 = 1 = x^0$.

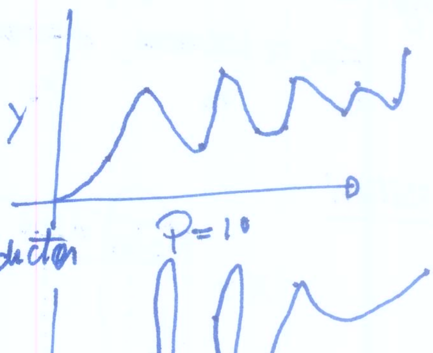
$\rightsquigarrow$ The predictor then becomes $\hat{y}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$

It is: it still fits the linear regression model but in a new **feature space** with additional features that are deterministic functions of our observations

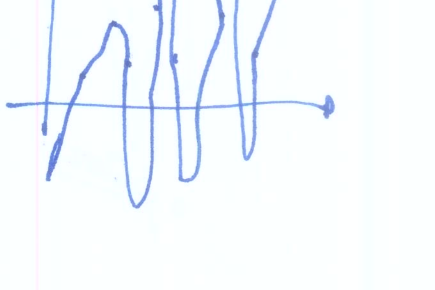
Applying the mse estimator $\hat{y}(x) = \sum_{i=1}^{10} \theta_i x^i$

~~of~~ Overfitting

In our polinomial fits for 11 data points, a more complex model $\hat{y}(x)$ a $p(x)$ with degree 10 and 11 coef. fits the data perfectly

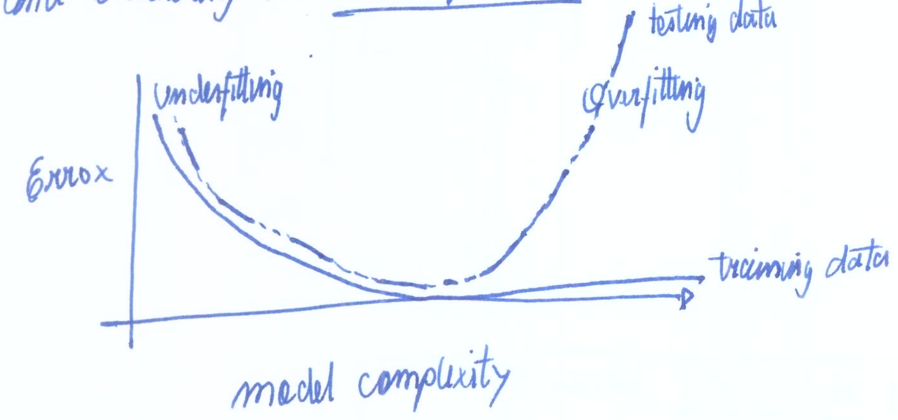


but it does not look like a good predictor



we fact an **overfit** to the data

We can see the $\boxed{\text{generalization}}$ or test error simply by gathering more data and evaluating the cost function



Continuing Regression with multiple variables

$\overset{x_1}{\text{size house}}$, $\overset{x_2}{\text{\# bedrooms}}$, $\overset{x_3}{\text{floors}}$, $\overset{x_4}{\text{years}}$, $\overset{y}{\text{Price}}$

$x^{(i)}$ training example.

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

~> Previously we had a model/hypothesis: $\hat{y}(\mathbf{x}) = \theta_0 + \theta_1 x$.

~> Let's update or upgrade our model

$$\hat{y}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_4 x_4.$$

<u>Example</u> $\hat{y}(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3x_3 \underbrace{- 2x_4}_{\text{each year decreases de price.}}$

for convenience let's define $\underbrace{x_0 = 1}_{\text{zeroth feature always with the vale (1).}}$.

Then $\underbrace{x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}}_{\text{0-index}} \in \mathbb{R}^{n+1}$ $\underbrace{\theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix}}_{\text{0-index vector}} \in \mathbb{R}^{n+1}$

$h_\theta(x) = \hat{y}(x)$

$\hat{y}(x) = \Theta_0 \overset{\rightarrow 1}{x_0} + \Theta_1 x_1 \dots \Theta_n x_n$

$= \Theta^T x$

$$\underbrace{\begin{bmatrix}\Theta_0 & \dots & \Theta_n\end{bmatrix}}_{\Theta^T}\underbrace{\begin{bmatrix}x_0 \\ \vdots \\ x_n\end{bmatrix}}_{x}$$

$\boxed{\Theta^T x}$ ..

This is called multivariate linear regression

multivariate: multiple variables/features

×

How do we do the GD with multiple variables/features?

Hypothesis $\hat{y}(x) = \Theta^T x = \Theta_0 \overset{\rightarrow 1}{x_0} + \Theta_1 x_1 \dots \Theta_n x_n$.

Params: $(\Theta_0, \dots, \Theta_n)$. let's think it as a vector $\overset{\text{vector}}{\vec{\Theta}} \in \mathbb{R}^{n+1}$

Cost function $J(\Theta_0, \dots, \Theta_n) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}(x^{(i)}) - y^{(i)}\right)^2$.

$J(\vec{\Theta})$.

Gradient Desent Repeat

$$\Theta_j \leftarrow \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\vec{\Theta})$$

update at the same time for all $j \in \{1, \dots, n\}$.

How to perform the partial derivative?

GD for $n \geq 1$. Repeat {

$$\Theta_j \leftarrow \Theta_j - \alpha \cdot \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}(x^{(i)} - y^{(i)}\right) x_j^{(i)}$$

Only thing that changes

$x_0^{(i)} = 1$ by definition

}

Gradient Descent in practice 1: feature scaling

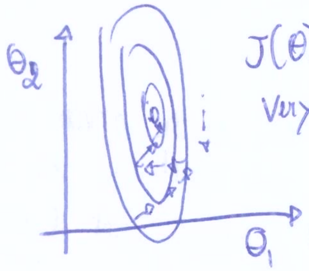
~> what do we do if we have some features dominating the others?

~> we need to scale the features in order GD can converge more quickly.

We need to make sure the features are in the same range of values.

Example x_1 = size (0-2000 m^2)

x_2 = nbr of bedrooms (1-5)



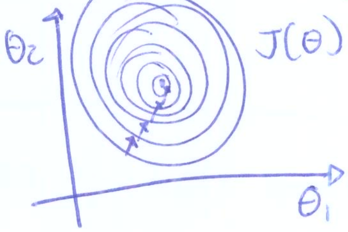
J(θ). Very tall and skinny ovals.

(ignoring θ_0)

if we run GD in here, it may take a while before converging

A nice workaround here would be to normalize the data then the contours would look like more circles

$x_1' = \frac{\text{size } (m^2)}{2000}$ $x_2' = \frac{\#\text{bedrooms}}{5.}$



$$\begin{cases} 0 \leq x_1 \leq 1 \\ 0 \leq x_2 \leq 1. \end{cases}$$

Feature scaling ~> Get every feature into approximately a $\underbrace{-1 \leq x_i \leq 1}_{\text{desired but not firmly fixed}}$ range

~> x_0 is already in the range ($x_0 = 1$)

~> $0 \leq x_1 \leq 3$ (is ok)

$-2 \leq x_2 \leq 4$ (is ok)

but

$-20 \leq x_3 \leq 40$ ✗

$-0.0001 \leq x_4 \leq 0.0001$ ✗.

~> How to do it? training data.

In addition to scale normalization, sometimes we perform a step further and do mean normalization

~> Replace x_i with $x_i - \mu_i$ for zero meaning the features

~> Do not change $x_0 = 1$.

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

if we do that, we will end up with values close enough to

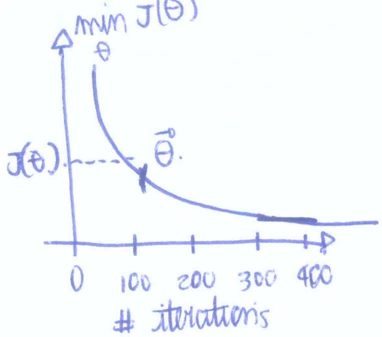
$-0.5 \le x_1 \le 0.5, \quad -0.5 \le x_2 \le 0.5$.

General rule $x_i \leftarrow \frac{x_i - \mu_i}{s_i}$ where μ_i is the mean of x_i in training set and s_i is the range of values in traing set $\underbrace{(max_i - min_i)}$.

sometimes it will interesting to use the std of feature (i)

$$x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i} \quad \left\} \begin{array}{l} z\text{-norm} \\ z\text{-score} \end{array} \right. \quad \boxed{\text{Data standardization}}$$

How do we know GD is working correctly and how to choose the learning rate α.

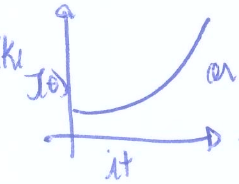


what you expect.

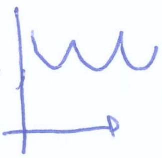
- $J(\theta)$ should decrease after iteration
- # of iterations vary according to the application

An automatic convergence would be possible, for instance if $J(\theta)$ in one iteration is only slightly lower (10^{-3} for instance) than $J(\theta)$ in the previous iteration. But this is difficult.

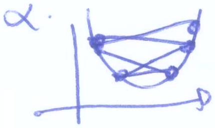
if you have something like



or



then choose a smaller α.



Reason of increase: big α overshoots the minimum.

Two general observations:

1. for small enough α, $J(\theta)$ should decrease on every iteration...
2. for small enough α or too small α, GD can be slow to converge.

But... how to choose α?

try... 0.001, 0.01, 0.1, 1, ... always plotting $J(\theta) \times$ # iterations

or

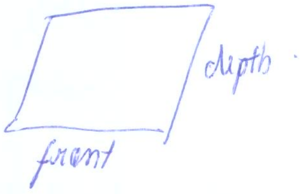
0.001, 3x0.001, 0.01, 3x0.01, ..., ...

or

make it linked to the nbr of iteration $\frac{\alpha}{\text{iteration}}$.

Features and Polynomial Regression

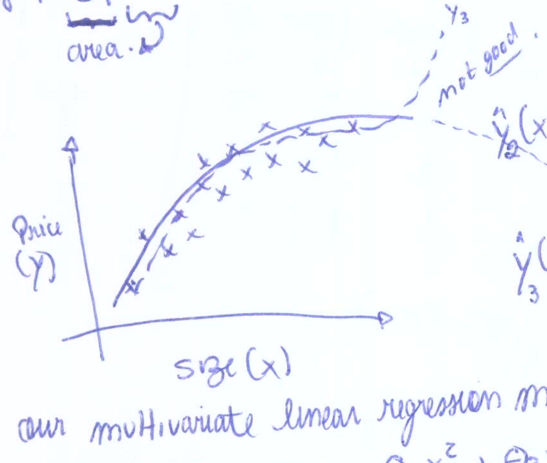
Housing prices $\hat{y}(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$



we can define a new feature $X = \text{front} \times \text{depth}$ (area)

then $\hat{y}(x) = \Theta_0 + \underbrace{\Theta_1 x}_{\text{area}}$.

Polynomial Regression



$\hat{y}_2(x) = \Theta_0 + \Theta_1 x + \Theta_2 x^2$.

or

$\hat{y}_3(x) = \Theta_0 + \Theta_1 x + \Theta_2 x^2 + \Theta_3 x^3$.

for using our multivariate linear regression model, we just define new features

$$\hat{y}(x) = \Theta_0 + \Theta_1 x + \Theta_2 x^2 + \Theta_3 x^3$$

$$\Downarrow$$

$\hat{y}(x) = \Theta_0 + \Theta_1 x + \Theta_2 x + \Theta_3 x$ where

$x_1 = \text{size}$, $x_2 = \text{size}^2$ and $x_3 = \text{size}^3$.

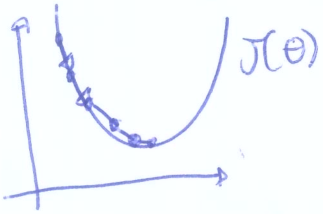
$$\left\{\begin{array}{l} \text{size}: 1 - 10^3 \\ \text{size}^2: 1 - 10^6 \\ \text{size}^3: 1 - 10^9 \end{array}\right.$$

$\boxed{\text{Note}}$: if we do this, feature ~~set~~ scaling is $\boxed{\text{paramount}}$

→ How to choose the features ? There are some algorithms for <u>that</u>.

The Normal Equations

Gradient Descent



The <u>Normal Equation</u> is a method for solving for θ <u>analytically</u> (one <u>step</u>).

$$\theta \in \mathbb{R}^{n+1} \quad \underset{\text{minimize}}{J(\theta_0 \ldots \theta_m)} = \frac{1}{2m} \sum_{i=1}^{m} \hat{y}(x^{(i)}) - y^{(i)})^2$$

as we want to minimize $J(\theta)$, we solve for

$$\frac{\partial J(\theta)}{\partial \theta_j} = \ldots \text{ set to } 0$$

(derive and equal to zero) for every (j).

solving for $\theta_0, \theta_1, \ldots, \theta_n$.

<u>For example</u>: for a given problem, put the feature vector in a matrix (X) and the outcomes (target) in a vector (y).

$$X = \begin{array}{c} \begin{array}{ccccc} & \text{area} & \text{bedrooms} & \text{floors} & \text{age} \end{array} \\ \left[\begin{array}{ccccc} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1400 & 2 & 3 & 40 \\ 1 & a & b & c & d \\ 1 & e & f & g & h \end{array}\right] \end{array} \qquad y = \begin{array}{c} \text{price (K\$)} \\ \left[\begin{array}{c} 460 \\ 230 \\ 500 \\ 200. \end{array}\right] \end{array}$$

$$\mathbb{R}^{M \times (n+1)} \qquad\qquad \mathbb{R}^{m}$$

$$\vec{\theta} = (X^T X)^{-1} X^T \overset{\Downarrow}{y}.$$ gives the <u>min</u>.

General Case

m examples $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$, n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

~> we will design what we call a design matrix

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

$\mathbb{R}^{M \times (n+1)}$

what does it mean $\vec{\theta} = (X^TX)^{-1} X^T y$.

1 $(X^TX)^{-1}$ is inverse of matrix X^TX.

~> when using Normal Equations, feature scaling is not necessary!

So, what should we use, GD or Normal Equation?

GD	Normal Equation
1 Need to choose α. ↓ 2 Needs many iterations	1 No α. ↑ 2 No iterations
1 works well with a high number of features n ↑	1 Needs to compute $(X^TX)^{-1}$ ↓ 2 slow if n is very large.

** $X^TX \in \mathbb{R}^{n \times n}$, therefore normally it costs $O(n^3)$ for inverting on most inverse implementations

what is large?

- n in hundreds, go with Normal Equation
- n in thousands (< 5K), OK with Normal Eq.
- n > 10K difinitely go with GD or some alternatives.

finally the normal equation will not work for more complex problems such as classification, that's why GD is important and should be always thought as one option.

Normal Equations and non invertibility.

⇝ when computing $\vec{\Theta} = (X^T X)^{-1} X^T y$, what if $(X^T X)^{-1}$ is non-invertible/singular/degenerate?

⇝ R and Octave and Matlab have workarounds (robust) inverse functions called pseudo-inve
that does the right thing.

⇝ normally there are two main causes for degeneration:

① Redundant features (linearly dependent)

↳ solution: _dim. reduction
- feature deletion/selection

Example x_1 = size in m^2. 1Km = 1000 meters

x_2 = size in Km^2

so $x_1 = (1000)^2 \cdot x_2$

② Too many features (# examples ⓜ ⋘ # features ⓝ)

↳ solution - dim reduction
- dim/feature selection/deletion
- Regularization

Example $\begin{cases} m = 10 \text{ examples} \\ n = 100 \text{ dimensions} \end{cases}$ $\overset{\text{params}}{\Theta} \in \mathbb{R}^{101}$

—— x ——

Organization of the class on August 14th, 2013

slide 9, 11, 12, 13, 14, 15, 16, 17, 18, ⑩.

Logistic Regression and Classification

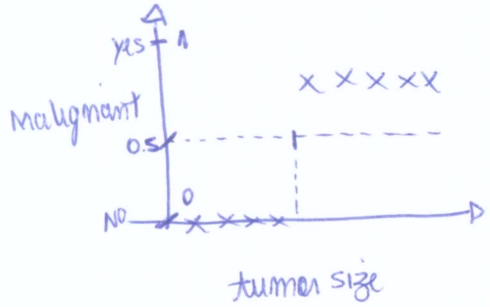
1. Our outcome now is discrete-valued.

2. ~~LR~~ LogR is one of the most used learning algorithms.

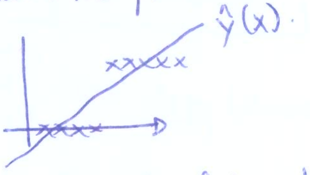
Some classification problems: spam × non spam (e-mail)
fraudulent × non-fraudulent (transactions)
tumor malign × benign
going blind × not going blind (DR retinal). diab. retinopathy analysis

$y \in \{0, 1\}$ positive class (1); neg. class (0), absence of something

binary classif. problem.



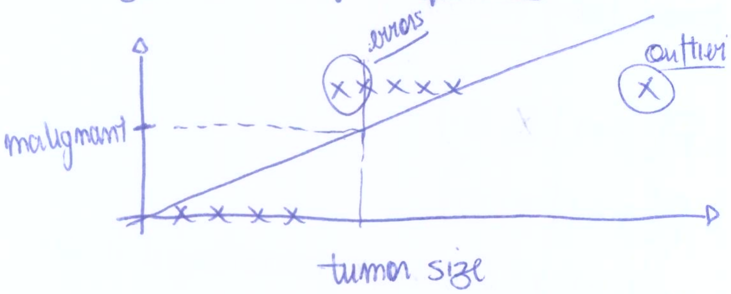
what if we approach the problem as a linear regression one?
$\hat{y}(x) = \theta^T x$.



Then we could threshold classifiers output at $\hat{y}(x)$ @ 0.5:

$$\begin{cases} \text{if } \hat{y}(x) \geqslant 1/2 \text{ predict } \textcircled{1} \\ \text{otherwise predict } \textcircled{0}. \end{cases}$$

But let's change the problem:



Often linear regression ~~is~~ isn't a good idea for classification.

⤳ Another problem with ~~LR~~ ~~LogR~~ linear regression is that even with training examples with $y \in \{0, 1\}$ it can output $\hat{y}(x) > 1$ or $\hat{y}(x) < 0$.

LogR solves this problem such that $0 \leq \hat{y}(x) \leq 1$ **always**.
⤷ we will use it as a classif. algorithm. The name is historical.

Hypothesis Representation

LogR model want $0 \le \hat{y}(x) \le 1$.

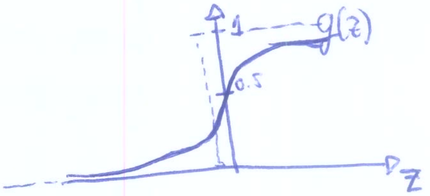
when we had LR, $\hat{y}(x) = \vec{\theta}^T x$, for LogR, we change it to $\hat{y}(x) = g(\vec{\theta}^T x)$,

linear regression

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

sigmoid or logistic function

Synonyms.



By doing that, my new predictor will be

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

Now we need to fit parameters for θ.

Interpretation of hypothesis Output

$\hat{y}(\cancel{x})$ = estimated prob that $y=1$ on input x.

Example $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

suppose my outcome is

$y(x) = 0.7$

Tell patient that 70% chance of tumor being malignant.

Mathmatically, we have $\hat{y}(x) = P(y=1 \mid \cancel{X}; \theta)$

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

prob. of $y=1$ given x, parameterized by θ.

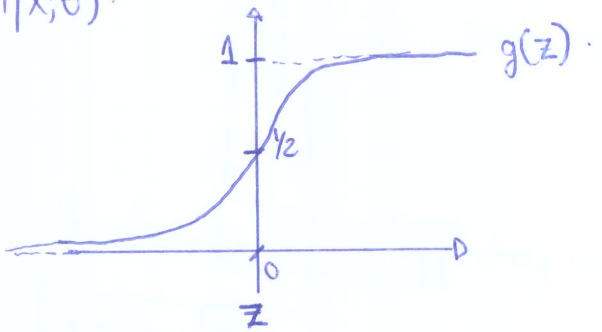
Properties $\begin{cases} P(y=0 \mid x;\theta) + P(y=1 \mid x;\theta) = 1. \\ P(y=0 \mid x;\theta) = 1 - P(y=1 \mid x;\theta). \end{cases}$

* y must take values in $\{0, 1\}$ binary value only.

Decision Boundary

LogR $\hat{y}(x) = g(\vec{\Theta}^T x)$ → $P(y=1|x;\theta)$.

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

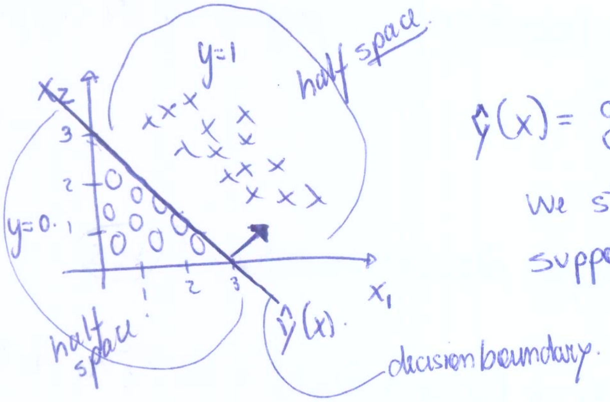


$\begin{cases} \text{Suppose predict } y=1 \text{ if } \hat{y}(x) \geq 0.5 \\ y=0, \text{ otherwise.} \end{cases}$

if we look at the <u>sigmoid</u> plot, we see $g(z) \geq 0.5$ when $z \geq 0$.

then given our predictor $\hat{y}(x) = g(\vec{\Theta}^T x)$, it will be ≥ 0.5 when $\underbrace{\vec{\Theta}^T x}_{z} \geq 0$.

Suppose we have training set



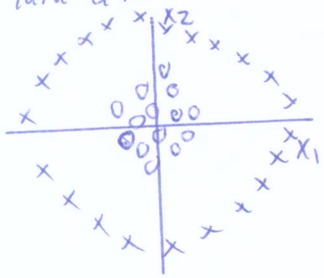
$\hat{y}(x) = g(\Theta_0 + \Theta_1 x_1 + \Theta_2 x_2)$.

We still don't know how to choose Θs but suppose $\begin{cases} \Theta_0 = -3 \\ \Theta_1 = 1 \\ \Theta_2 = 1. \end{cases}$ $\vec{\Theta} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$.

→ $\boxed{\text{Predict}}$ $y=1$ if $\underbrace{-3 + x_1 + x_2}_{\vec{\Theta}^T x} \geq 0$ rewritting $\boxed{x_1 + x_2 \geq 3}$.
→ $y(x) = 0.5$ <u>exactly</u>.

→ The decision boundary is a <u>property</u> of the hypothesis including the params $\Theta_0 \cdots \Theta_2$.
and not of the <u>data set</u>.

Let's take a look at a more complex example



→ how can we fit LogR params to solve this problem?

→ We can transform the feature space to higher order just like we did with <u>polynomials</u>.

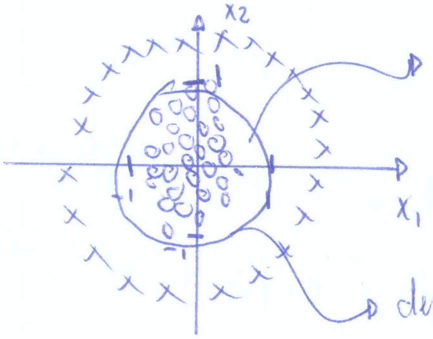
$$\hat{y}(x) = g(\Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_1^2 + \Theta_4 x_2^2)$$

added features.

Suppose we have found $\vec{\Theta} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$.

My predictor will predict $\boxed{y=1}$ if $-1 + x_1^2 + x_2^2 \geq 0$

$$\boxed{x_1^2 + x_2^2 \geq 1}$$



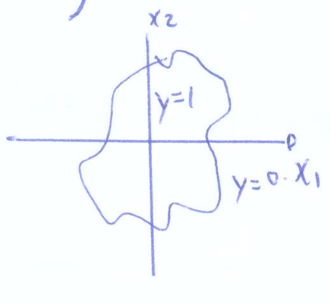
→ Equation of the circle $\boxed{x_1^2 + x_2^2 \neq 1}$.

→ decision ~~[illegible]~~ boundary.

→ Once again, the decision boundary is a property of the hypothesis/model and not of the dataset

→ Can we have even more complex decision boundaries? yes.

$$\hat{y}(x) = g\left(\Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_1^2 + \Theta_4 x_1^2 x_2 + \Theta_5 \cdot x_1^2 \cdot x_2^2 + \Theta_6 x_1^3 x_2 \cdot \ldots\right)$$



But, how do we $\boxed{\text{automatically}}$ fit the params for a model/predictor?

for (logR).

training set $\left\{\left(x^{(1)},y^{(1)}\right),\dots,\left(x^{(m)},y^{(m)}\right)\right\}$

(m) examples $x\in\begin{bmatrix}x_0\\ \vdots\\ x_n\end{bmatrix}_{\mathbb{R}^{n+1}}$, $x_0=1$, $\underbrace{y\in\{0,1\}}_{\text{classif. problem.}}$

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

hypothesis

Question: how to choose parameters Θ?

Recall that in the linear Regression hypothesis, we had $J(\theta)=\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(\hat{y}(x^{(i)})-y^{(i)}\right)^2$

for (logR), let's call $\frac{1}{2}\left(\hat{y}(x^{(i)})-y^{(i)}\right)^2$ as $\boxed{\text{cost}\left(y(x^{(i)}),y\right)}$

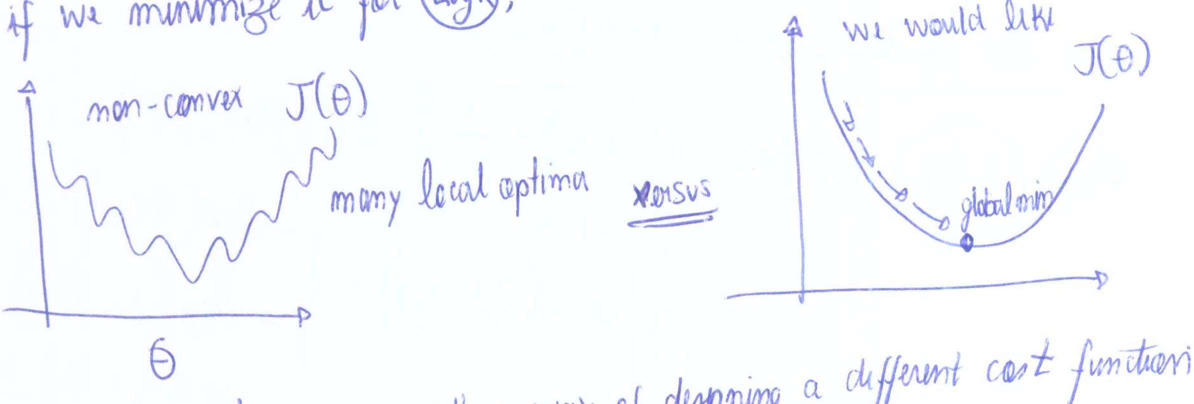
So $$\text{Cost}\left(\hat{y}(x^{(i)}),y^{(i)}\right)=\frac{1}{2}\cdot\left(\hat{y}(x^{(i)})-y^{(i)}\right)^2$$

Simplifying,

for LogR has non linearity $\frac{1}{1-e^{-\theta^T x}}$ complicated non linear function

$$\text{Cost}\left(\hat{y}(x),y\right)=\frac{1}{2}\cdot\left(\hat{y}(x)-y\right)^2$$

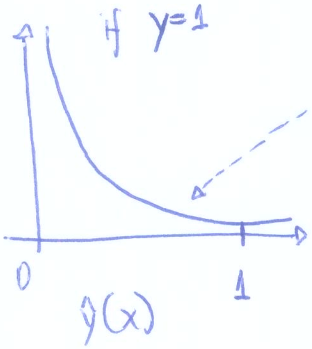
this cost function works fine for (LR) but we are focused on (LogR) and if we minimize it for (LogR), it will be clear it isn't a convex function



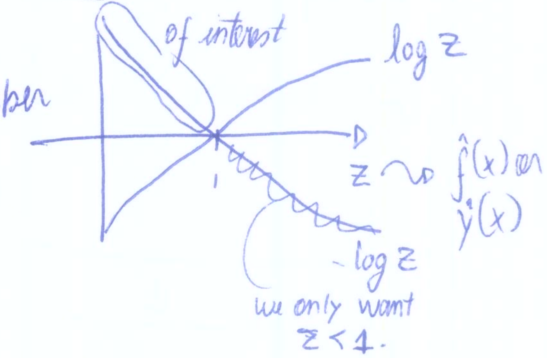
So we need to come up with a way of designing a different cost function that is convex.

Logistic Regression Cost function

$$Cost\left(\hat{f}(x), y\right) = Cost\left(\hat{y}(x), y\right) = \begin{cases} -\log\left(\hat{y}(x)\right) & \text{if } y=1 \\ -\log\left(1-\hat{y}(x)\right) & \text{otherwise } (y=0) \end{cases}$$



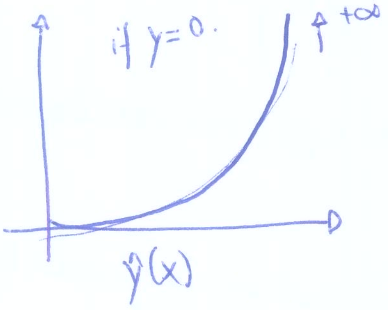
Remember



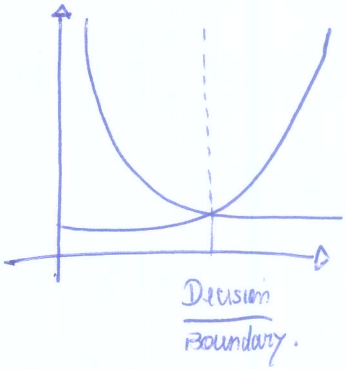
$\leadsto$ Cost = 0 if $y = 1$, and prediction is $\hat{y}(x) = 1$

But as $\hat{y}(x) \leadsto 0$, Cost $\leadsto \infty$.

we penalize the learning alg. by a **very** large cost as it commits mistakes (go further away from the corrects value/output).



putting **together**



simplified Cost function and Gradient Descent

LReg cost function $J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(\hat{y}(x^{(i)}, y^{(i)}\right)$

where

$$Cost\left(\hat{y}(x), y\right) = \begin{cases} -\log\left(\hat{y}(x)\right) & \text{if } y=1 \\ -\log\left(1-\hat{y}(x)\right) & \text{otherwise} \end{cases}$$

$y \in \{0, 1\}$ binary problem

Let's compress the two cases in one.

$$\text{Cost}\left(\hat{y}(x), y\right) = -y \log\left(\hat{y}(x)\right) - (1-y)\log\left(1-\hat{y}(x)\right)).$$

so now, we can design our cost function as

$\Downarrow$

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

$$= \boxed{-\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \cdot \log\left(\hat{y}(x^{(i)})\right) + (1-y^{(i)}) \cdot \log\left(1-\hat{y}(x^{(i)})\right)\right]}.$$

we choose it because it can derived from statists using principle of **MLE** max. likelihood estim. and it is convex.

- To fit params $\vec{\theta}$: $\min_{\vec{\theta}} J(\vec{\theta})$.
- To make a prediction for a new x, Output $\hat{y}(x) = \frac{1}{1+e^{-\vec{\theta}^T x}}$ which means $P(y=1 \mid x; \underbrace{\vec{\theta}}_{\text{learned}})$

Using GD for minimizing $J(\vec{\theta})$.

want $\min_{\theta} J(\vec{\theta})$ Repeat{

$$\theta_j \leftarrow \theta_j - \alpha \cdot \underbrace{\frac{\partial}{\partial \theta_j} J(\theta)}_{**}$$

$$** \quad \frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \cdot \sum_{i=1}^{m} \left(\hat{y}(x^{(i)}) - y^{(i)}\right) \cdot x_j^{(i)}$$

what? it look exactly the same as before for LR.

But the change is in $\hat{y}(x)$

before $\hat{y}(x) = \theta^T x$.

now $\hat{y}(x) \overset{\text{now}}{=} \frac{1}{1+e^{-\theta^T x}}$

smooth

In a vectorized form :

$$\begin{bmatrix}\theta_0\\ \vdots\\ \theta_m\end{bmatrix}_{m\times 1} \leftarrow \begin{bmatrix}\theta_0\\ \vdots\\ \theta_m\end{bmatrix}_{m\times 1} - \alpha \left(\begin{bmatrix} Z(X) \end{bmatrix}_{m\times m} - \begin{bmatrix} Y_1\\ \vdots \\ Y_m\end{bmatrix}_{m\times 1} * \right)_{m\times 1} \begin{bmatrix} X \end{bmatrix}^{T}_{m\times m}$$

$m \times m$

$\boxed{\text{No}}$ some adjustments are necessary .

$$\begin{bmatrix} f(A) \end{bmatrix}_{m\times m} \begin{bmatrix} B \end{bmatrix}_{m\times 1} = \begin{bmatrix} C \end{bmatrix}_{m\times 1}$$

$$\begin{bmatrix} f(A) \end{bmatrix}_{m\times 1} - \begin{bmatrix} Y \end{bmatrix}_{m\times 1} = \begin{bmatrix} C \end{bmatrix}_{m\times 1}$$

Logistic Regression - advanced optimization

So far, we have seen the opt. alg Gradient Descent and with it, we have a cost function $J(\vec{\theta})$ and want $\min_{\theta}(J(\theta))$.

Given $\vec{\theta}$, we compute ① $J(\theta)$

② $\frac{\partial}{\partial \theta_j} J(\theta) \quad \forall j = 0, \dots, n_{features}$.

and (GD) does: Repeat:

$$\{ \quad \theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \quad \}.$$

but this is one alternative. If we show how to compute $J(\theta)$ and $\frac{\partial}{\partial \theta_j} J(\theta)$, we can use more sophisticadded solutions such as

- Conjugate Gradient
- BFGS
- L-BFGS.

Normally, such alternatives ① Do not need α (learning rate) manually selected.

② often faster than GD.

however, they are more complex.

Example on how to use them:

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

of course $\theta = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$ minimizes it and that's what we want to find.

$$\frac{\partial J(\theta)}{\partial \theta_1} = 2(\theta_1 - 5)$$

$$\frac{\partial J(\theta)}{\partial \theta_2} = 2(\theta_2 - 5)$$

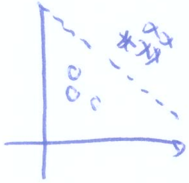
Normally, all we need to do is to write code for computing $J(\theta)$ and its derivatives.

Logistic Regression on multi-class problems - One vs All.

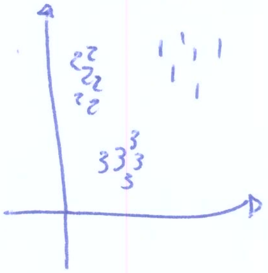
Example: digits, faces {profile left, profile righ, frontal}, age-group estimation, foldering/tagging photos emails,

foldering {family, business, gym, ⋮} $y \in \{1, \dots, k\}$ classes.

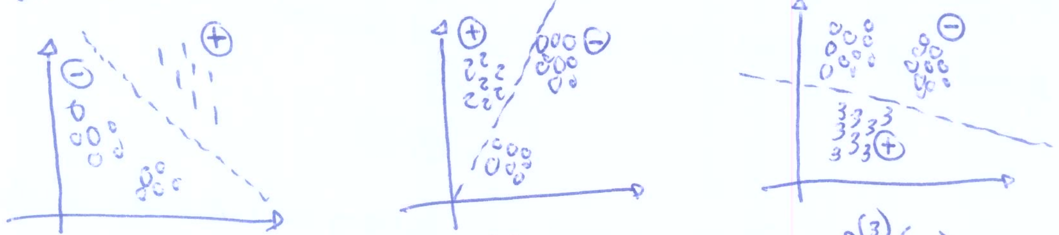
Binary Classif



Multi-class



Let's do it one-vs-all (OVA): first we transform our problem on 3 problems.



$f_\theta^{(1)}(x)$ $f_\theta^{(2)}(x)$ $f_\theta^{(3)}(x)$

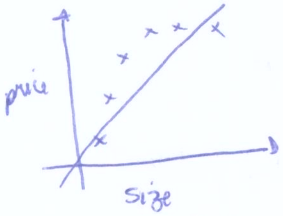
What we did was ~~$f(x)$~~ $f_\theta^{(i)}(x) = P(y=i|x;\theta)$ $i \in \{1, \dots, 3\}$.

Basically, we train a LogR classifier $f_\theta^{(i)}(x)$ for each class i to predict $P(y=i)$. For a new input x, for predicting, we select the class that maximizes $\boxed{\max_i \hat{f}_\theta^{(i)}(x)}$

Regularization and the problem of overfitting

Regularization will allow us ~~do~~ to diminish overfitting problems.

Lets return to the problem of housing

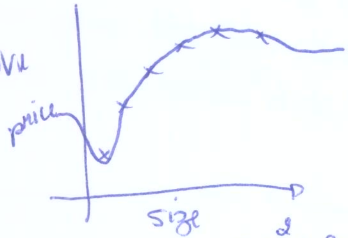


$\theta_0 + \theta_1 x.$

~> underfit, high bias } algorithm has a very strong bias or pre-conception that the housing prices change linearly with ~~the~~ the size even the data shows the contrary

both mean the model is not fitting the data very well.

alternatively, we could have



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

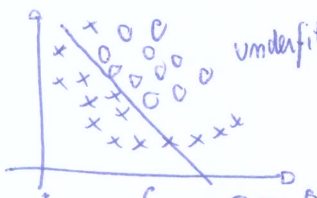
Overfit, "high variance".

the model fitted the data perfectly and it is to just to it and the hypothesis is too variable since there is not enough data to prove it

overfitting : if we have too many features, the learned hypothesis may fit ~~the~~ the training set very well $\left(J(\theta) = \frac{1}{2m}\sum_{i=1}^{m} f_\theta(x^{(i)}) - y^{(i)})^2 \approx 0\right)$ but $\boxed{\text{fail}}$ to generalize to new examples (predict prices on new data).

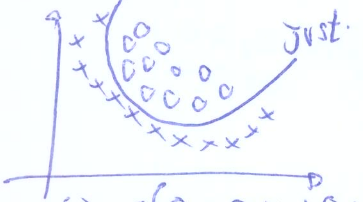
The same thing can happen with logistic regression as well.

Overfit / high variance.

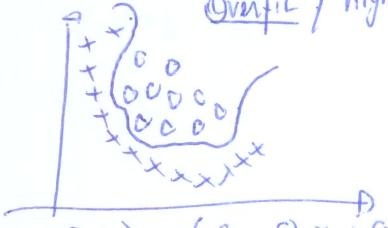


$f_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

g = sigmoid function

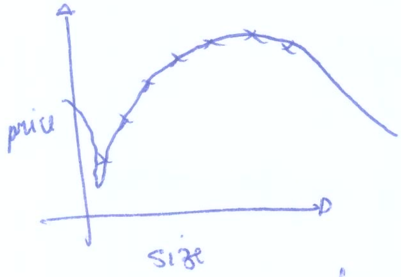


$f_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 \boxed{x_1^2} + \theta_4 \boxed{x_2^2} + \theta_5 \boxed{x_1 x_2}).$



$f_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 \boxed{x_1^2 x_2} \theta_4 \boxed{x_1^2 x_2^2} + \theta_5 \boxed{x_1^2 x_2^3} \ldots).$

Addressing Overfitting



→ Plotting is one way but it wouldn't work for high-dimensional data

Two main options to deal with it

1 Reduce number of features → manually which features to keep.
→ model selection algorithm.
automatic alg. that select which features to keep/throw away.

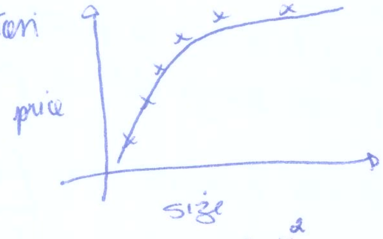
* however, sometimes this means you are throwing away importan info about the problem.

2 Regularization

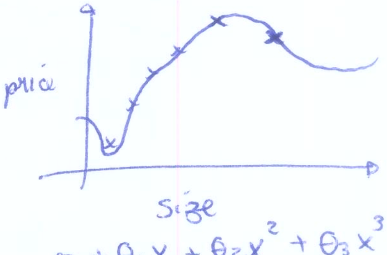
→ Keep ALL the features but reduce magnitude/values/importance of parameters θ_j. works well ~~with~~ ~~many~~ when we have lots of features, ~~and~~ each of which contributing a bit for predicting y.

Regularization - Cost function

Intuition



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



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

Suppose we penalize and make θ_3 and θ_4 really small

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} \left(f_\theta(x^{(i)}) - y^{(i)} \right)^2 + \underbrace{\boxed{1000\,\theta_3^2 + 1000\,\theta_4^2}}_{\text{high penalization}}$$

this would give $\theta_3 \approx 0$
$\theta_4 \approx 0$

Regularization Cost function

Besides the intuition, here's the general idea:

~ small values for parameters $\theta_0, \theta_1, \dots \theta_n$ } - Simpler hypothesis
- less prone to overfitting.

Let's see the housing example

- features: $x_1, x_2, \dots, x_{100}$
- params: $\theta_0, \theta_1, \dots, \theta_{100} \in \mathbb{R}^{101}$

it is difficult to pick one or some to penalize beforehand. If it was easy, it would be the same as selecting the less important features!

So, ~~x~~ here's what we do:

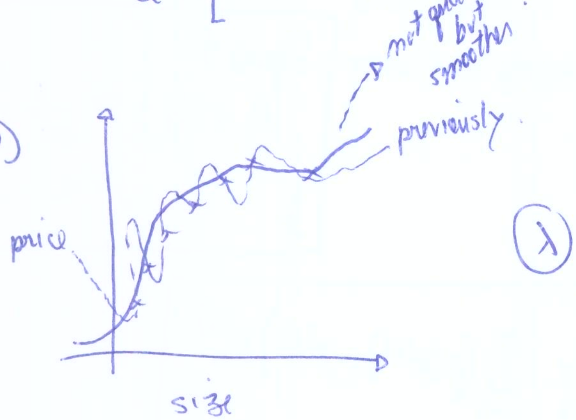
$$J(\theta) = \frac{1}{2m}\left[\overbrace{\sum_{i=1}^{m}\left(f_\theta(x^{(i)}) - y^{(i)}\right)^2}^{\text{good fitting}} + \overbrace{\boxed{\lambda \sum_{i=1}^{m} \theta_j^2}}^{\text{Regularization}}\right]$$

Keep the params small.

do not penalize θ_0

Regularization parameter

$\min_\theta J(\theta)$

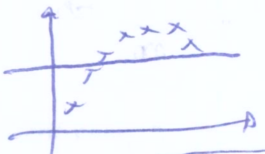


λ controls the tradeoff between good fitting to training data vs Keep the parameters small.

and with both we want to keep the hypothesis simple and avoid ~~of~~ overfitting.

Question: what if λ is too high (eg. $\lambda = 10^{10}$) in a regularized linear regression?

↳ all θs except θ_0 will tend to zero and the model will be biased



$$\theta_0 + \boxed{\theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4} \to 0 \text{ zero.}$$

therefore $f_\theta(x) = \theta_0$.

} underfitting

too strong bias/pre-conception the a straight line in θ_0 will fit well in spite of data on the contrary.

⇝ Be <u>carefull</u> when choosing $\boxed{\lambda}$.

Regularized Linear Regression

So far, for linear regression, we have seen <u>GD</u> and <u>Normal Equation</u> for fitting the parameters.

Here's the optimization function for regularized linear regression

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

$$\Downarrow$$

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

Previously, the $\boxed{GD}$ for minimizing this was

$$\boxed{\begin{array}{l}\text{Repeat } \{\\ \quad \theta_j \leftarrow \theta_j - \alpha\cdot\frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)})-y^{(i)}\right)x_j^{(i)}\\ \qquad\qquad j=0,\dots,m.\\ \}\end{array}}$$

if we separate θ_0:

$$\boxed{\begin{array}{l}\text{Repeat } \{\\ \quad \theta_0 \leftarrow \theta_0 - \frac{\alpha}{m}\cdot\sum_{i=1}^{m}\left(f(x^{(i)})-y^{(i)}\right)x_0^{(i)}\\ \quad \theta_j \leftarrow \theta_j - \frac{\alpha}{m}\cdot\sum_{i=1}^{m}f(x^{(i)})-y^{(i)})x_j^{(i)}.\\ \}\end{array}}$$

$$\Downarrow$$

$$\begin{array}{l}\text{Repeat } \{\\ \quad \theta_0 \leftarrow \theta_0 - \alpha\cdot\overbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)})-y^{(i)}\right)\cdot x_0^{(i)}}^{\frac{\partial J(\theta)}{\partial\theta_0}.}\\ \quad \theta_j \leftarrow \theta_j - \alpha\underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}\left(f(x^{(i)})-y^{(i)}\right)\cdot x_j^{(i)} + \underbrace{\frac{\lambda}{m}\cdot\theta_j}_{\text{regularization}}\right]}_{\frac{\partial J(\theta)}{\partial\theta_j}\text{regularized}}\\ \}\end{array}$$

We can regroup things and <u>then</u>

$$\Theta_j \leftarrow \Theta_j - \alpha \cdot \left[\frac{1}{m} \sum_{i=1}^{m} \left(f(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} + \frac{\lambda}{m} \cdot \Theta_j \right]$$

$$\Downarrow$$

$$\Theta_j \leftarrow \Theta_j \underbrace{\left(1 - \frac{\overset{\text{small}}{\alpha} \overset{\text{small}}{\lambda}}{\underset{\text{large}}{m}} \right)}_{< 1} - \frac{\alpha}{m} \cdot \sum_{i=1}^{m} \left(f(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)}.$$

for instance $1 - \frac{\alpha \lambda}{m} < 1$

$1 - 0.01 = \boxed{0.99 \cdot \Theta_j}$ shrinks Θ_j a <u>little bit</u>.

what about <u>normal equations</u>?

$$X = \underbrace{\begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}}_{m+(n+1)} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad \mathbb{R}^m.$$

$(x^{(1)})^T \to$ one training example.

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

$$\Downarrow$$

$\vec{\Theta} = (X^T X)^{-1} \cdot X^T y$ without Regularization

for regularization;

$$\vec{\Theta} = \left(X^T X + \lambda \underbrace{\begin{bmatrix} 0 & & \overset{\text{zeros}}{0} \\ & 1 & \\ \underset{\text{zeros}}{0} & & 1 \\ & & & 1 \end{bmatrix}}_{(n+1)x(n+1)} \right)^{-1} \cdot X^T y.$$

<u>Example</u> $m = 2 \curvearrowright \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

<u>what about non-invertibility now?</u> / <u>(Advanced)</u>

$$\vec{\Theta} = \overbrace{(X^T X)}^{\text{non-invertible / singular.}}{}^{-1} \cdot X^T y$$

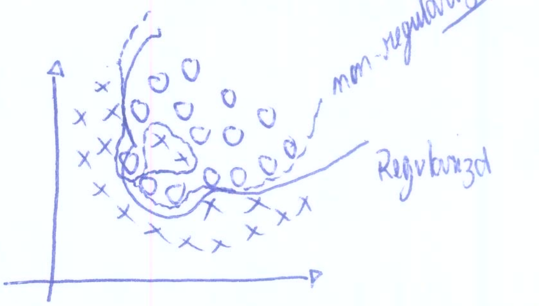
Suppose $\underset{\#\text{exam}}{m} \leqslant \underset{\#\text{features}}{n}$

$\boxed{\text{but}}$ if $\lambda > 0$, $\vec{\Theta} = \left(X^T X + \lambda \begin{bmatrix} 0 & & \\ & 1 & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} \cdot X^T y.$

we'll have no singularity any<u>more</u>. (why) some features will decrease magnitude and simplify model.

Regularized Logistic Regression

RegL can ~~[illegible]~~ also be prone to overfitting:



$$\hat{f}_\theta(x) = \overset{\text{sigmoid}}{g}\left(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 \cdot x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots\right)$$

$\Downarrow$

Cost function

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

introducing Regularization

We just add $\underbrace{+ \frac{\lambda}{2m} \cdot \sum_{j=1}^{\overset{\#\text{ dimensions}}{n}} \theta_j^2}_{A}$ for $\theta_1, \dots \theta_n$.

How to implement this?

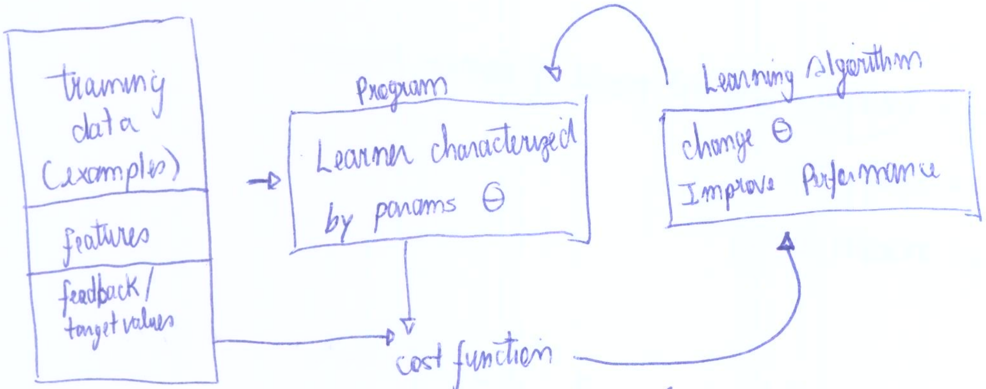
Gradient Descent

```
Repeat {
    $\theta_0 \leftarrow \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(f(x^{(i)}) - y^{(i)}\right) \cdot x_0^{(i)}$
    $\theta_j \leftarrow \theta_j - \alpha \underbrace{\left[\frac{1}{m} \sum_{i=1}^{m} \left(f(x^{(i)} - y^{(i)}\right) \cdot x_j^{(i)} \oplus \overset{\text{regularization}}{\boxed{\frac{\lambda}{m} \theta_j}}\right]}_{\frac{\partial J(\theta)}{\partial \theta_j}}$
        $j \in \{1, \dots, n\}$
}
```

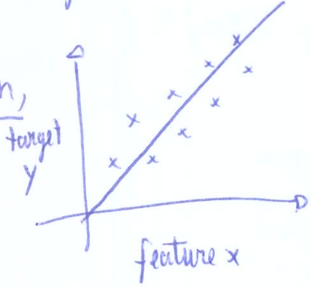
remember $f_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$

Wrapping up what we had so far

Notation
features x
targets y
predictions $\hat{y}$
params θ



In the case of linear regression,



"Predictor"
Evaluate line
$r = \theta_0 + \theta_1 x_1$

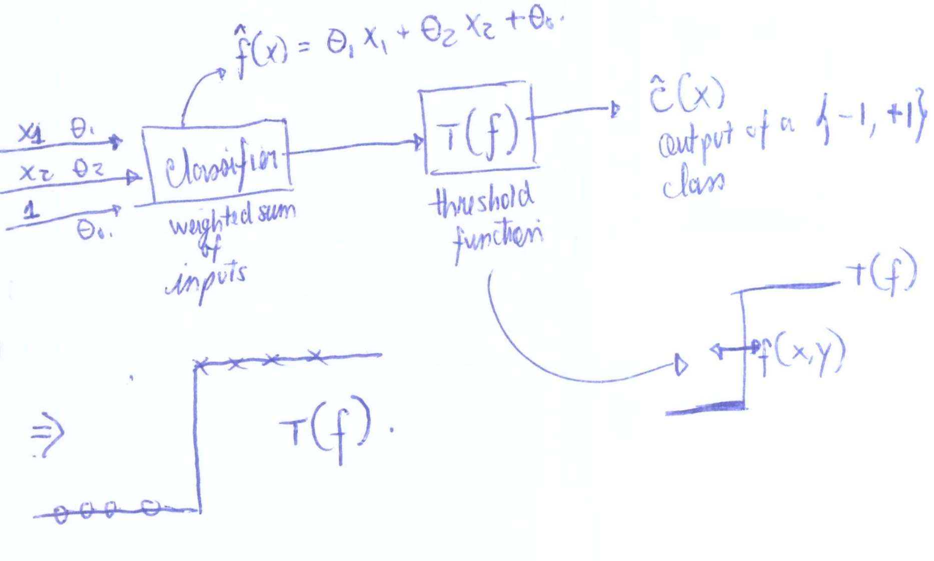
$$\boxed{\hat{y}(x) = \theta_0 + \theta_1 x_1}$$

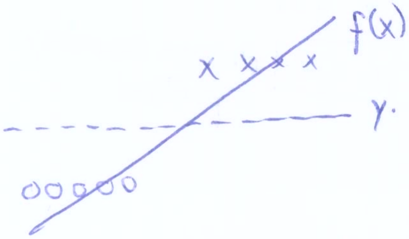
$y \in \mathbb{R}$.

while with classification, we seek to $\boxed{\text{predict}}$ a discrete value/target y.

Now that we already saw linear regression and logistic regression (classifier) let's see a similar linear algorithm for classification called perceptron.

Perceptron with 2 features





→ Perceptron is a linear classifier

→ The $\boxed{W_s}$ will be the weights (Θ_s earlier).

W_0 is the intercept $w_0 = 1$ iniatialy.

A perceptron calculates ② quantities
- ① a weighted sum of the input features
- ② a threshold on this sum by the Ⓣ function.

The perceptron is a (simple) artificial model of human neurons

weights → synapses

threshold → neuron firing

Perceptron operation :

● $\hat{f}(x_1, \dots x_n) = \begin{cases} 1 & \text{if } \theta_1 x_1 + \dots + \theta_n x_n > 0 \\ 0 & \text{otherwise (can also be defined as } -1) \end{cases}$

$\vec{\Theta}$ → weight vector $\in \mathbb{R}^{n+1}$

$\vec{X}$ → feature vector $\in \mathbb{R}^{n+1}$.

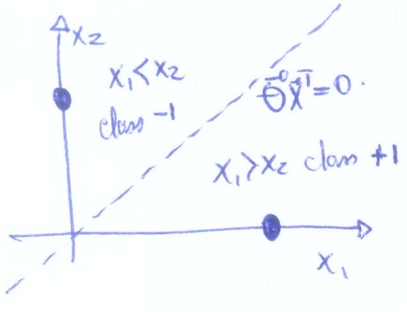
$$\Theta_0 x_0 + \Theta_1 x_1 + \dots \Theta_n x_n = \underbrace{\vec{\Theta} \vec{X}^T}_{\text{vector inner product}}$$

The perceptron represents a hyperplane decision surface in n-dimensional space { 2d line ↘ 3d ↘ plan

The equation of the hyperplane is $\underline{\Theta \vec{X}^T = 0}$

This is the equation por points in X-space that are on the boundary.

Example



Suppose $\vec{\Theta} = (\Theta_1 = 1, \Theta_2 = -1, \Theta_0 = 0)$.

$\vec{\Theta} \vec{X}^T \Rightarrow 1x_1 + (-1 . x_2) + 0(x_0) = 0$.

$x_1 - x_2 = 0$

$\boxed{x_1 = x_2}$ decision boundary equation

* A data set is ~~linearly~~ separable by a learner if there is some instana of that learner that correctly predicts all data points

* The learnear is linearly separable if:
- Can separate the two classes using a straight line if $X \in \mathbb{R}^2$ or with a hyperplane ~~if~~ if $X \in \mathbb{R}^n$.

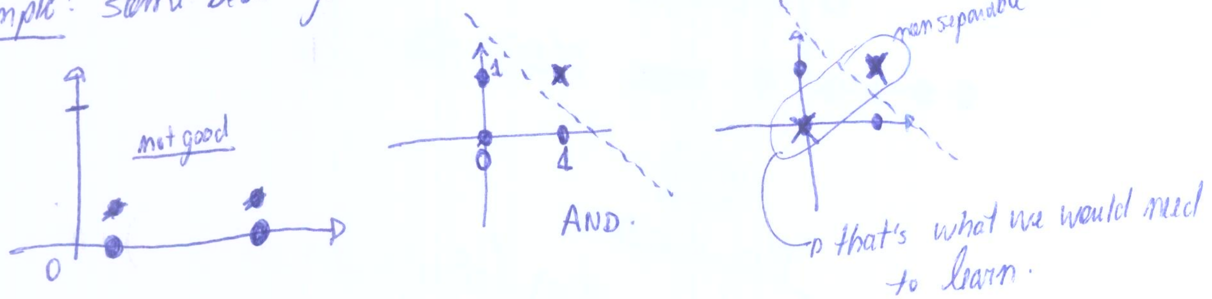
Class overlap { Common in practice
Same observation values possible for both classes (benign/malign cells look similar).

what to do? { - Increase number of features.
- Increase model complexity.

Representational Power of perceptrons

→ what mappings a perceptron can represent perfectly?
well, it is a linear classifiers, so it can represent any mapping that is linearly separable

Example: some boolen functions (AND) but no XOR



Remember Effects of dimensionality

1. Data are increasingly separable in high dimension – Is this a good thing?

Good { - Separation is easier in higher dimensions (for fixed M) ↳ nbR examples.
- Increase nbR of features, and even a linear classifier would do the trick!

Bad { - training vs test error; overfitting; bias vs variance
- Increasingly complex decision boundaries can eventually get all training data right but it doesn't mean necessarily good for testing data ⇒ lacks generalization

How to update weights ?

for each data point $x^{(j)}$:

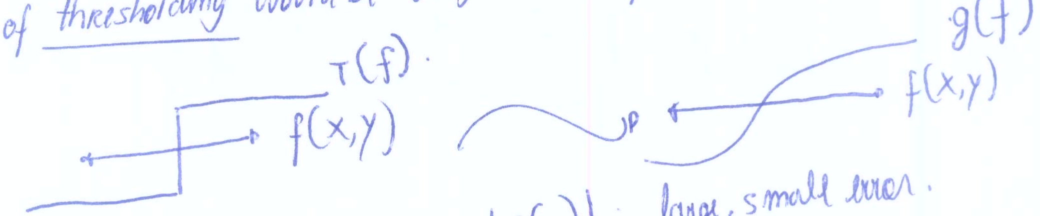
$f(x^{(j)}) \leftarrow T(\vec{w} * x^{(j)})$

$\vec{w} \leftarrow w + \alpha\left(f(x^{(j)}) - y^{(j)}\right) \cdot x^{(j)}$ gradient-like step

→ if $f(x^{(j)})$ correctly predicted : no update.

otherwise : update weights.

Another way of thresholding would be using smoother functions

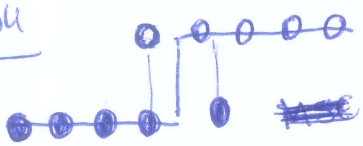


sigmoid for instance

if we are far from decision boundary, $|f(.)|$ is large, small error.

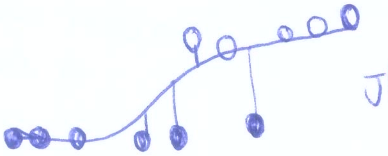
nearby the boundary: $(f(.))$ near 1/2, large error.

Example



$J(\theta) = \frac{2}{10}$ normal thresholding

while



$J(\theta) = (0^2 + 1^2 + .2^2 + .25^2 + \dots)/10$.

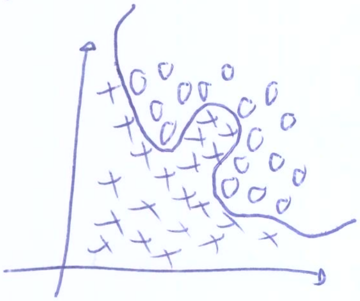
~~Sum~~ Summary

- linear classifer ⟺ perceptron
- visualizing the decision boundary : useful but not always possible
- Measuring quality : Sum of squared errors (SSE).
- Learning the weights of a linear classifier reduces to an optimization problem. For SSE we can do gradient descent but there are others

Neural Networks – Non-linear hypotheses

- Quite old idea
- Out of favor for sometime but in the hype again for several recent results such as Deep Belief Networks (Deep learning) and results such as image nets
- why do we need yet another learning algorithm?

Consider a problem



→ we could use LogReg such as

$$\hat{f}(x)=g(\theta_0+\theta_1x_1+\theta_2x_2+\theta_3\underline{x_1x_2}+\theta_4\underline{\cancel{x_1}x_2^2}+\dots)$$

poly terms

→ normally it works ok for a few features (2,3) but not as good for many features (how to derive polynomials there?).

if we have 100 features and want to just include 2nd order polynomials, we would have

$$\hat{f}(x)=g(\cancel{\theta_0}+\theta_1x_1+\theta_2x_2+\dots+\theta_{100}x_{100}+\theta_{101}x_1^2+\theta_{102}x_1x_2,\dots,\theta_Kx_1x_{100}+\theta_{K+1}x_2x_3+\dots)$$

$\approx$ 5K features. $\Theta(n^2)$ $\approx \frac{n^2}{2}$.

orig. feature space

→ If we want 3rd order polys, we would have $\Theta(n^3)$.
For 100 features ≈ 170K features!

→ In practice, we have problems with thousands or millions of features in the original ~~space~~ feature space.

Neurons and the brain

Origins:
- algorithms that try to mimic the brain.
- widely used in the 80's and early 90s. Popularity went down in late 90s.
- Recent hype due to state-of-the-art results in many applications.
- the reason is that now we have enough computational power to design and use large-scale neural ~~net~~ networks

expensive to run

The "one learning algorithm" hypothesis

→ Some people believe the brain does all of its amazing things based on one learning algorithm.

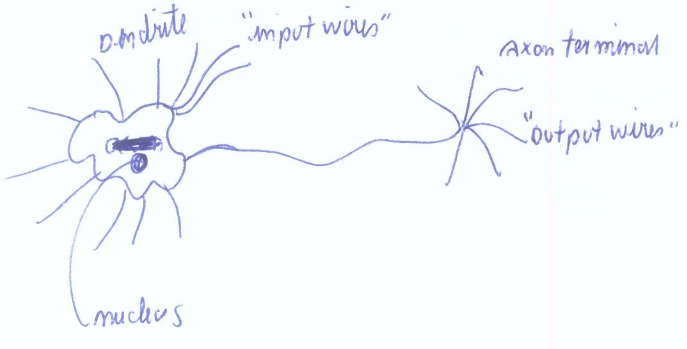
→ Auditory cortex learns to ~~se~~ hear/listen but researchers have shown that cutting the connection of Auditory cortex to the ear and rewiring it to the eye leads to a learning process in which the auditory cortex learns to see!

neuro-rewiring experiments.

Examples : - Brainport experiment : Camera for low-res grayscale image on top of the head and a wire to conect each pixel to our tongue (low voltage → dark pixel, high voltage → brigh pixel, and we can "see" through our tongue.

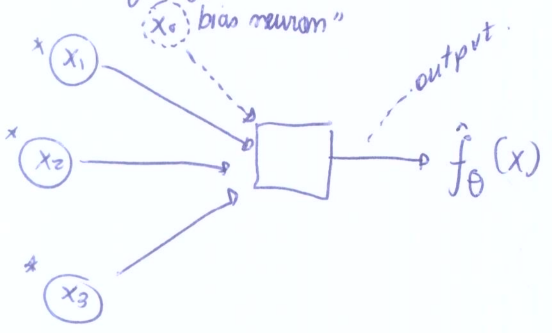
- Human echolocation (sonar) : snap fingers. Boy that had his eyeballs removed learned to navigate using this "rewiring" technique.
- Implanting a 3rd eye in a frog will lead him to actually learn how to use it.

Neural Networks - Representation



Computatiolly speaking, we can think of a neuron as a computational unit that receives a set of inputs, performs some computation and outputs some signals through the axons to other neurons.

We are going to model this as a logistic unit



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

$$\vec{x} = \begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix} \quad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix}.$$

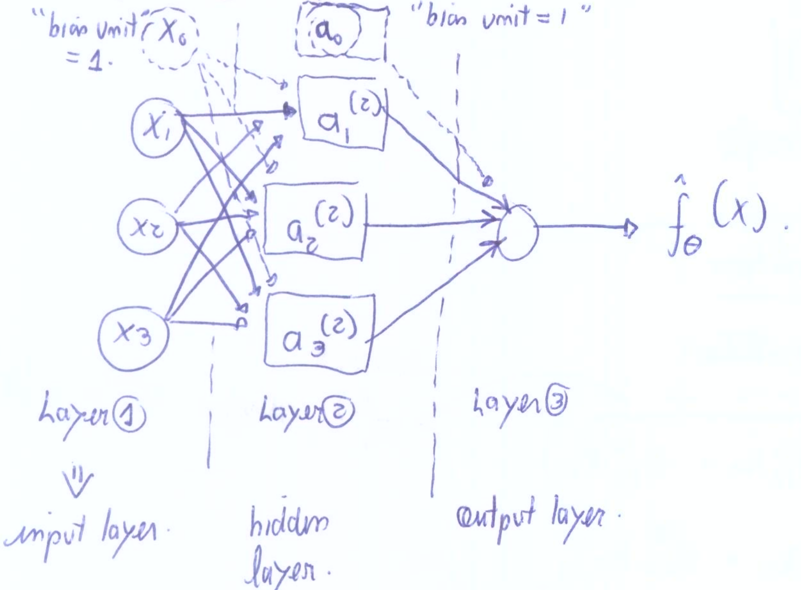
* => inputs.

x_0 => bias neuron always equal to ①.

In NN terminology the sigmoid/logistic activation function and the $\vec{\theta}$ parameters are known as weights.

represents

Generalizing, a NN is just a group of several neurons.



because it is not ⓧ or ⓨ (input/output) therefore the term "hidden".

↝ We may have NNs with several hidden layers.

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

$\vec{\Theta}^{(j)}$ = matrix of weights controlling function mapping from layer (j) to layer (j+1).

Here's the computation:

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

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

$$a_3^{(3)} = g\left(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3\right).$$

$$f_\theta(x) = a_1^{(3)} = g\left(\theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{3}^{(2)} a_3^{(2)}\right).$$

$\Theta^{(1)}$ is a matrix mapping layer (1) to (2), so $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$.

↝ If network has (s_j) units in layer (j), and (s_{j+1}) in layer (j+1), then $\Theta^{(j)}$ will have dimensions $(s_{j+1}) \times (s_j + 1)$.

↝ with this, we have $\boxed{\hat{f}_{\Theta}(x)}$ our classifier.

---

model Representation, intuition and calculations

forward propagation: vectorized implementation

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

let's call $z_1^{(2)}$, therefore $a_1^{(2)} = g\left(z_1^{(2)}\right)$.

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3\right) \quad z_2^{(2)}$$

$$a_3^{(2)} = g\left(\underbrace{\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3}\right) \quad z_3^{(2)}.$$

$\boxed{\Theta^{(1)} \cdot \vec{X}}$ or $\boxed{\vec{\Theta}^{(1)} \vec{X}}$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_3 \end{bmatrix} \; ; \quad Z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \; ;$$

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

$$\underbrace{a^{(2)}}_{\to \mathbb{R}^3} = g(\underbrace{z^{(2)}}_{\to \mathbb{R}^3}).$$

g function applies <u>sigmoid</u> function on each $z^{(i)}$ element-wise. Given all elements in $z^{(i)}$ (e.g. ~~in~~ in $\mathbb{R}^3$), we apply g on each individually.

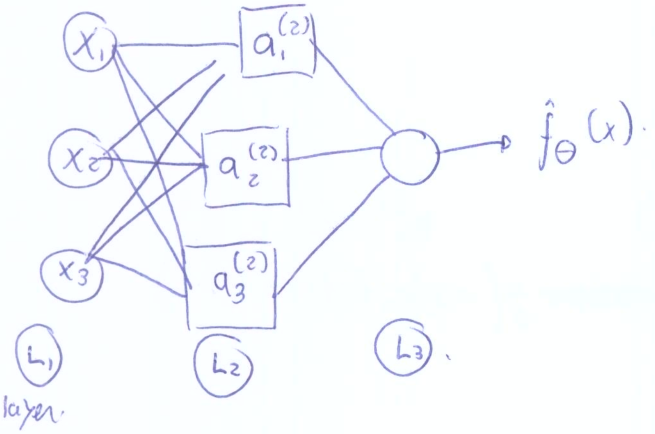
Let's define $a^{(1)} = \vec{x}$., <u>therefore</u> $z^{(2)} = \vec{\Theta}^{(1)} . \vec{x}$ now becomes $\boxed{z^{(2)} = \vec{\Theta}^{(1)} . a^{(1)}.}$

Now we add $\underset{\text{bias unit}}{\boxed{a_0^{(2)} = 1}}$

$$\left.\begin{aligned} &\text{so that } a^{(2)} \in \mathbb{R}^4 \\ &\underline{\text{and}} \\ &z^{(3)} = \vec{\Theta}^{(2)} . a^{(2)} \\ &\hat{f}_\Theta(x) = a^{(3)} = g(z^{(3)}). \end{aligned}\right\}$$

This algorithm is known as forward propagation.

Neural Network learning its own features



* Suppose we don't see the left part <u>of it</u>.

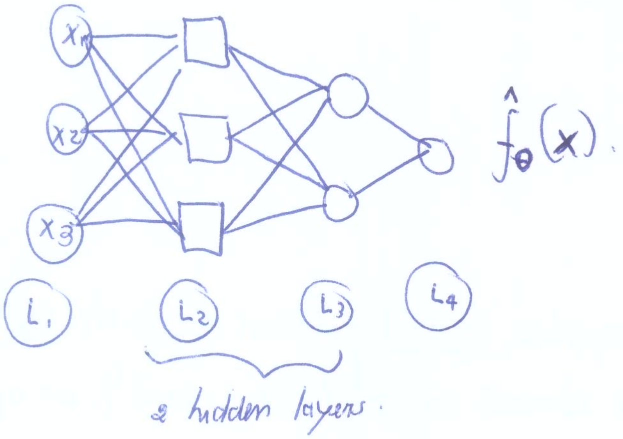
if we do that, we end up having a logistic Regression like situation

$$\hat{f}_\Theta(x) = g\left(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}\right).$$

The nice thing here is the the features a_i are <u>learned</u> from the <u>input</u>.

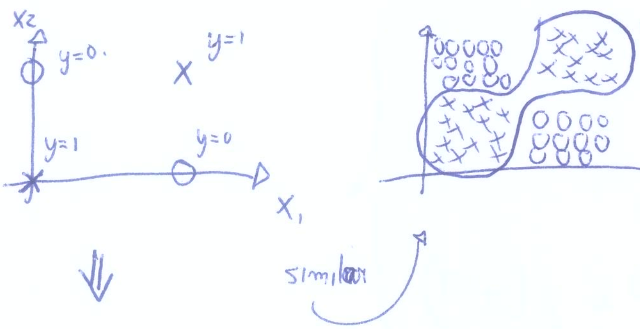
Observation : we can have other NN diagrams/architectures

↳ how NN ~~[illegible]~~ units connect to each other.



x

Let's return to ~~[illegible]~~ our logical operators example in which x_1 and x_2 are binary (0 or 1).



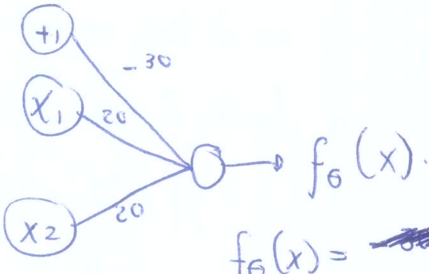
we want to learn a non-linear separator/classifier

$$\begin{cases} y = x_1 \text{ XOR } x_2 \\ y = \text{XNOR } x_2 \\ \boxed{\text{Not}(x_1 \text{ XOR } x_2)}. \end{cases}$$

Let's start with a simple example for logical AND.

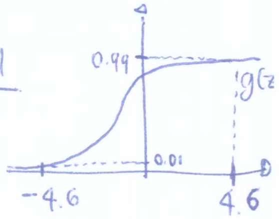
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2.$



$$f_\theta(x) = g(\overset{\theta_{10}^{(1)}}{-30} x_0 + \overset{\theta_{11}^{(1)}}{20} x_1 + \overset{\theta_{12}^{(1)}}{20} x_2)$$

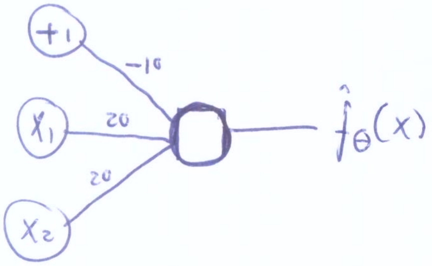
sigmoid



x_1	x_2	$\hat{f}_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$.

$$\boxed{f_\theta(x) \approx x_1 \boxed{\text{AND}} x_2}$$

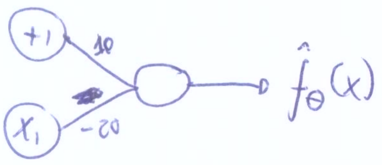
Example for (OR) function



x_1	x_2	$\hat{f}_\Theta(x)$
0	0	$g(-10) \approx 0$
0	1	$g(+10) \approx 1$
1	0	$g(+10) \approx 1$
1	1	$g(10) \approx 1$.

$$\hat{f}_\Theta(x) = g\left(-10x_0 + 20x_1 + 20x_2\right).$$

Example for NOT x_1



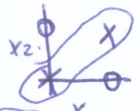
x_1	$f_\Theta(x)$
0	$g(10) \approx 1$.
1	$g(-10) \approx 0$.

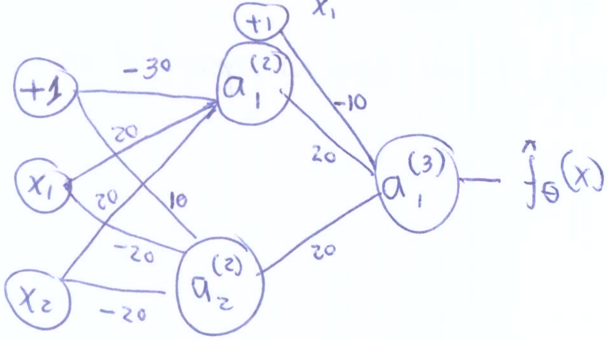
$$\hat{f}_\Theta(x) = g(10 - 20x_1)$$

Exercize: how to compute
→ (Not x_1) AND (Not x_2) ?

---

x_1 NOR x_2





x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$\hat{f}_\Theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1.

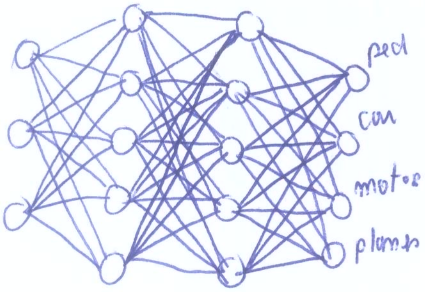
$a_1^{(2)}$ is using x_1 AND x_2 network
$a_2^{(2)}$ is using $\neg x_1$ and $\neg x_2$ or (not x_1) AND (not x_2).

* Each layer in the sequence can learn more complex functions ~~than~~ on top of its predecessors's output layer.

Multiclass Classification

Basically, it is an extension of the one vs all method.

$\boxed{\text{Pedestrian}}$ $\boxed{\text{Car}}$ $\boxed{\text{motorcycles}}$ $\boxed{\text{Planes}}$



$\hat{f}_\theta(x) \in \mathbb{R}^4.$

We want $\hat{f}_\theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ when pedestrian, $\hat{f}_\theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ when car, $\hat{f}_\theta(x) = \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ when motorcycles and $\hat{f}_\theta(x) = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$ when plane.

In the training set, $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \ldots, (x^{(m)}, y^{(m)})$

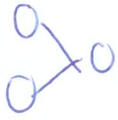
$y^{(i)}$ is going to be one of $\underset{\text{ped}}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}}, \underset{\text{car}}{\begin{bmatrix}0\\1\\0\\0\end{bmatrix}}, \underset{\text{motor}}{\begin{bmatrix}0\\0\\1\\0\end{bmatrix}}, \underset{\text{plane}}{\begin{bmatrix}0\\0\\0\\1\end{bmatrix}}$

~ previously, we have represented $y \in \{1,2,3,4\}$ (e.g., K-NN). Now we represent it in a binary form. So for ④ classes,

$$\underbrace{\hat{f}_\theta(x)}_{\to \mathbb{R}^4.} \approx \underbrace{y^{(i)}}_{\to \mathbb{R}^4}$$

Cost function for Neural Networks

~ Let's start with the classification problem



$\boxed{L=2}$.

$s_1 = 2$.

$s_2 = 1$

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}.$$

Ⓛ will be the number of layers in the network.

$\boxed{s_l}$ = number of units (excluding the "bias" unit) in layer Ⓛ.

In the binary classification, $y=0$ or $y=1$, therefore we will have $\overset{\mathbb{R}^1}{①}$ output unit. For a multiclass classification problem (K classes), $y \in \mathbb{R}^K$, therefore we will have Ⓚ output units.

Let's now define a $\boxed{\text{cost function}}$. Recall that for logistic regression, we had :

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log \hat{f}_\theta(x^{(i)}) + (1-y^{(i)}) . \log(1-\hat{f}_\theta(x^{(i)}))\right] + \underbrace{\frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2}_{\substack{\text{Regularization} \\ \theta_0 = 1}}$$

for a neural network, instead of having just one logistic regression output unit, we will have Ⓚ of them.

$$\hat{f}_\theta(x) \in \mathbb{R}^K \quad (\hat{f}_\theta(x))_i = \boxed{i^{th}} \text{ output.}$$

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

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

Wow ! How can we optimize such cost function to actually find our weights or parameters ?

Backpropagation Algorithm

Given an $J(\theta)$, we want to $\min_{\theta} J(\theta)$. For that we need to compute:

① $J(\theta)$

② $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$. ; recall $\Theta_{ij}^{(l)} \in \mathbb{R}$.

Suppose we only have one training example (X, y). Using the forward propagation, we have

$$a^{(1)} = X$$

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

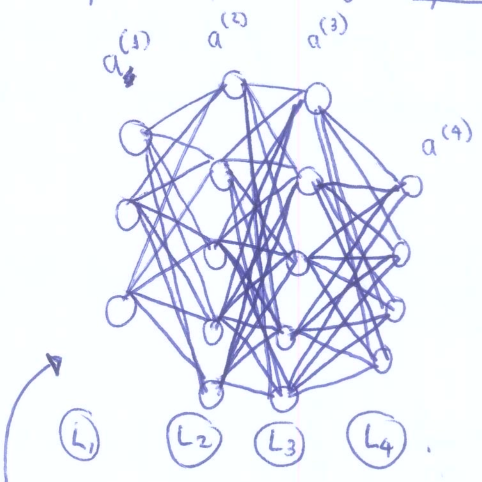
$$a^{(2)} = g(z^{(2)}) \quad \text{add } a_0^{(2)}$$

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

$$a^{(3)} = g(z^{(3)}) \quad \text{. add } a_0^{(3)}$$

$$z^{(4)} = \Theta^{(3)} a^{(3)}$$

$$a^{(4)} = \hat{f}_{\Theta}(x) = g(z^{(4)}).$$



$a^{(1)}$ activation values of input layer.

Gradient computation: Backpropagation algorithm

Intuition $\delta_j^{(l)}$ = "error" of node (j) in layer (l).

$a_j^{(l)} \Rightarrow$ activation of (jth) unit in layer (l).

Concretly, if we have the (NN) above, for each output unit (layer $L=4$),

$$\delta_j^{(4)} = \underbrace{\boxed{a_j^{(4)}}}_{(f_\Theta(x))_j} - y_j$$

if we vectorize it, we have $\vec{\delta}^{(4)} = \vec{a}^{(4)} - \vec{y}$. $\in \mathbb{R}^K$ where (K) is the number of output units in the (NN).

After that, we compute the (δ) terms for the earlier layers:

$$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \cdot \delta^{(4)} \; .* \; g'(z^{(3)}).$$

(.*) element-wise multiplication.

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} \; .* \; g'(z^{(2)}).$$

The derivative $g'(z^{(3)})$ is basically :

$$g'(z^{(3)}) = \underbrace{a^{(3)}}_{\text{vector}} .* \underbrace{(\overset{\text{vector of ones}}{\vec{1}} - a^{(3)})}_{\text{Vector}}.$$

$$g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)}).$$

→ There is no $\delta^{(1)}$ since they are the features we observe in our training set

→ The name back propagation comes from the fact we start in the output layer, calculate the error $\delta^{(4)}$ ~~the~~ there and go back to the 3rd layer where we compute $\delta^{(3)}$ and go back to layer ② and calculate $\delta^{(2)}$ in such a way it is propagating the error from layer $(l+1)$ to (l) and from (l) to $(l-1)$ and so on.

Finally $\boxed{\dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}}$ when ignoring the regularization term $(\lambda = 0)$.

Now, for a larger training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$, set

$\Delta_{ij}^{(l)} = 0 \quad \forall i, j, l.$

→ will be used as accumulators for computing $\dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.

```
For (i=1) to (m) {   // on each iteration, we work with one example.
    Set $a^{(1)} = x^{(i)}$
    Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, (L)$.
    Using $y^{(i)}$, compute $\boxed{\delta^{(L)} = a^{(L)} - y^{(i)}}$.   // computing error for this example for the output layer.
    Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$   used to compute the partial derivatives in the end $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$.
    $\boxed{\Delta_{ij}^{(l)} \leftarrow \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}}$   → //
}
```

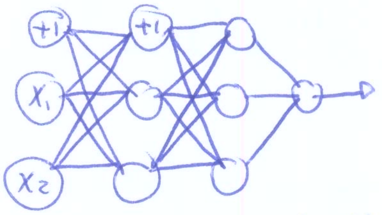
→ if we vectorize, $\Delta^{(l)} \leftarrow \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$.

$$\left.\begin{cases} D_{ij}^{(l)} \leftarrow \dfrac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ D_{ij}^{(l)} \leftarrow \dfrac{1}{m} . \Delta_{ij}^{(l)} & \text{if } j = 0 . \text{ (bias term case)} \end{cases}\right\} \text{ is exactly } \boxed{\dfrac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}}$$

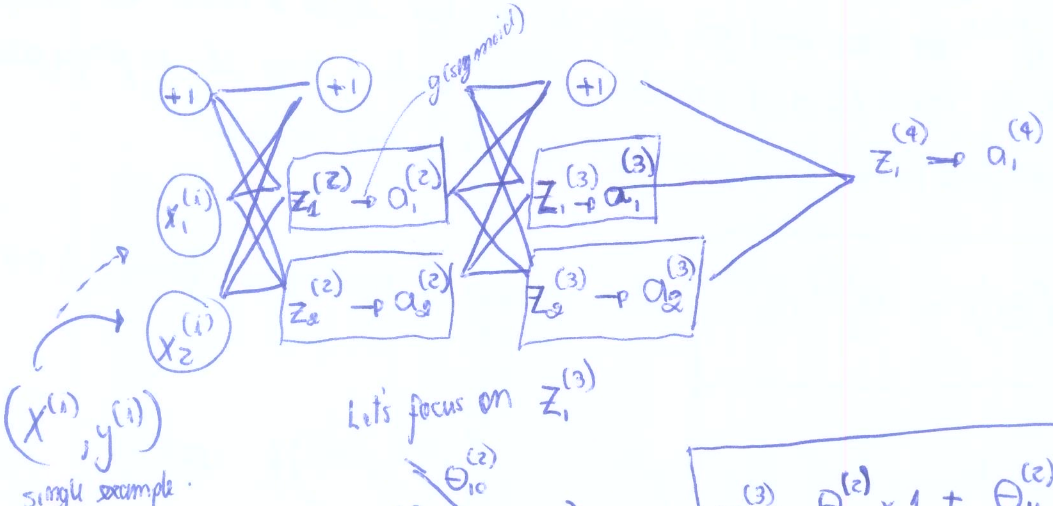
Backpropagation Intuition

mechanical steps.

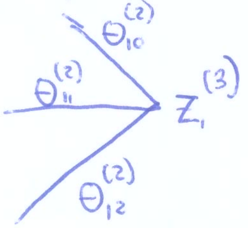
Let's see again forward propagation



2 units 2 un 2 un 1 unit
not counting bias


+1
$x_1^{(i)}$
$x_2^{(i)}$
+1
$z_1^{(2)} \to a_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
g (sigmoid)
+1
$z_1^{(3)} \to a_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$(x^{(i)}, y^{(i)})$
single example.

Let's focus on $z_1^{(3)}$



so $\boxed{z_1^{(3)} = \Theta_{10}^{(2)} \times 1 + \Theta_{11}^{(2)} \cdot a_1^{(2)} + \Theta_{12}^{(2)} a_1^{(2)}.}$

this is forward propagation.

What does Backpropagation do?

Recall its formulation

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log\left(f_\theta(x^{(i)})\right) + (1-y^{(i)}) \cdot \log\left(1 - f_\theta(x^{(i)})\right)\right] + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)}_{\text{Regularization.}}$$

focusing on a single example $(x^{(i)}, y^{(i)})$, the case of ① output unit and $\boxed{\lambda = 0}$

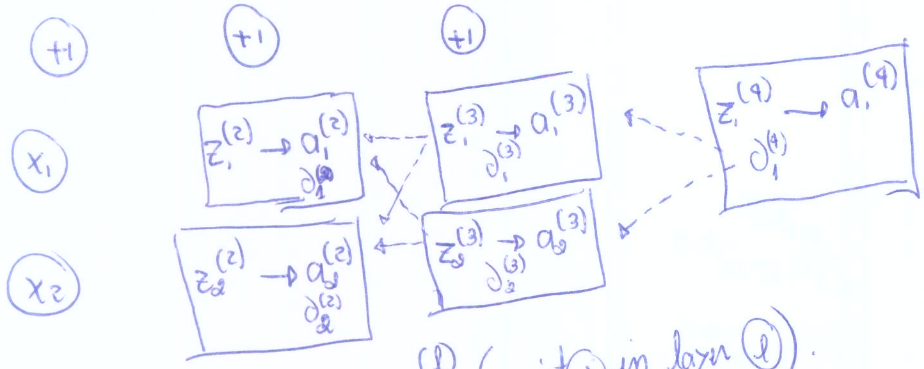
$$\text{Cost}(i) = y^{(i)} \log f_\theta(x^{(i)}) + (1-y^{(i)}) \log \underbrace{\left(f_\theta(x^{(i)})\right)^{*}}_{\text{the correct form is}}$$

* $\log(f_\theta(x^{(i)}))$. but should be $\underbrace{\log(1 - f_\theta(x^{(i)})}$

think of $\text{Cost}(i) \approx f_\theta(x^{(i)}) - y^{(i)})^2$.

How to the (NN) on example ⓘ ?

Forward Propagation


+1
+1
+1
x_1
x_2
$z_1^{(2)} \to a_1^{(2)}$
$\delta_1^{(2)}$
$z_2^{(2)} \to a_2^{(2)}$
$\delta_2^{(2)}$
$z_1^{(3)} \to a_1^{(3)}$
$\delta_1^{(3)}$
$z_2^{(3)} \to a_2^{(3)}$
$\delta_2^{(3)}$
$z_1^{(4)} \to a_1^{(4)}$
$\delta_1^{(4)}$

$\delta_j^{(\ell)}$ = error of cost for $a_j^{(\ell)}$ (unit j in layer ℓ).

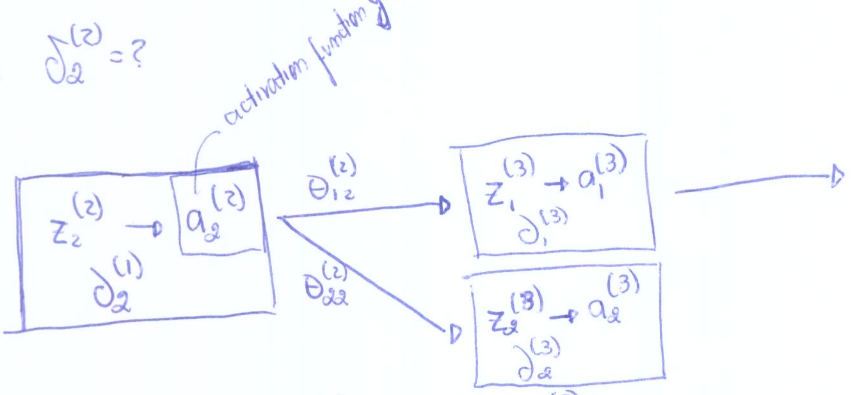
Formally, $\delta_j^{(\ell)} = \underset{\text{change here affects *}}{\frac{\partial}{\partial z_j^{(\ell)}}} \text{cost}(i)$ (for $j \geq 0$) where

$$\text{Cost}(i) = y^{(i)} \cdot \log \underbrace{f_\theta(x^{(i)})}_{\text{* here}} + (1 - y^{(i)}) \cdot \log\left(1 - \underbrace{f_\theta(x^{(i)})}_{\text{and * here}}\right).$$

Let's see it:

for $\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$ output.

how to get $\delta_2^{(2)} = ?$



$$\begin{cases} \delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}. \\ \text{and} \\ \delta_{(2)}^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)}. \end{cases}$$

Implementation Note

Example $s_1 = 10$ units

$s_2 = 10$

$s_3 = 1$ output unit

	derivatives
$\Theta^{(1)} \in \mathbb{R}^{10\times 11}$	$D^{(1)} \in \mathbb{R}^{10\times 11}$
$\Theta^{(2)} \in \mathbb{R}^{10\times 11}$	$D^{(2)} \in \mathbb{R}^{10\times 11}$
$\Theta^{(3)} \in \mathbb{R}^{1\times 11}$	$D^{(3)} \in \mathbb{R}^{10\times 11}$.

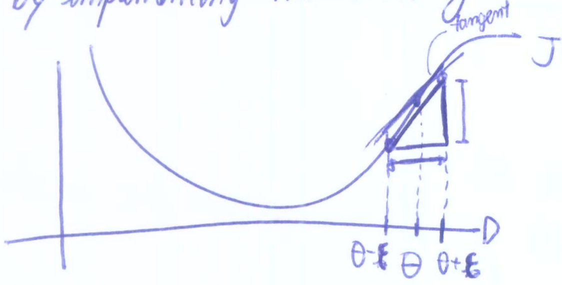
If we want to use this with more advanced ~~t~~ techniques of optimization, we may need to deal with them as ~~[illegible]~~ vectors instead of as matrices.

E.g., when optimizing in Octave.

Gradient Checking

How can we check the Backpropagation algorithm along with its optimization function for finding the parameters (e.g., gradient descent) are actually working?

⇝ We start by implementing numerical gradient checking.



$\theta \in \mathbb{R}$.

The slope of the line $\big(J(\theta+\varepsilon), (\theta+\varepsilon); J(\theta-\varepsilon), \theta-\varepsilon)\big)$ is an <u>approx</u> for the derivative of $J(\theta)$.

$$\text{slope} = \frac{\text{vertical height}}{\text{horizontal width}} = \frac{J(\theta+\varepsilon)-J(\theta-\varepsilon)}{2\varepsilon}.$$

Therefore $\dfrac{\partial}{\partial\theta} J(\theta) \approx \dfrac{J(\theta+\varepsilon)-J(\theta-\varepsilon)}{2\varepsilon}$ } two sided difference.

$\varepsilon = 10^{-4}$ (the smaller the ε, the closer to the actual derivative.

In a more general <u>case</u>, $\theta \in \mathbb{R}^n$ (e.g., θ is "unrolled" version of $\theta^{(1)}, \theta^{(2)}, \theta^{(3)}$)

<u>unrolled</u> matrix 2x2 $\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$.

$$\theta = [\theta_1, \theta_2, \dots, \theta_m]^T.$$

we can do:

$$\frac{\partial}{\partial\theta_1} J(\theta) \approx \frac{J(\boxed{\theta_1+\varepsilon}, \theta_2, \theta_3, \dots \theta_m) - J(\boxed{\theta_1-\varepsilon}, \theta_2, \dots, \theta_m)}{2\varepsilon}.$$

$$\frac{\partial}{\partial\theta_2} J(\theta) \approx \frac{J(\theta_1, \boxed{\theta_2+\varepsilon}, \theta_3, \dots \theta_m) - J(\theta_1, \boxed{\theta_2-\varepsilon}, \theta_3, \dots \theta_m)}{2\varepsilon}.$$

$$\frac{\partial}{\partial\theta_m} J(\theta) \approx \frac{J(\theta_1, \theta_2, \dots \boxed{\theta_m+\varepsilon}) - J(\theta_1, \theta_2, \dots, \boxed{\theta_n-\varepsilon})}{2\varepsilon}.$$

These equations allow us to numerically estimate the derivatives with respect to any parameter θ_i and then can be used to double check the derivatives calculated by backpropagation.

~> Backpropagation is very efficient compared to gradient numerical checking. Therefore we should use it just in one or a few iterations to ~~the~~ double check backpropagation results.

x

Random Initialization

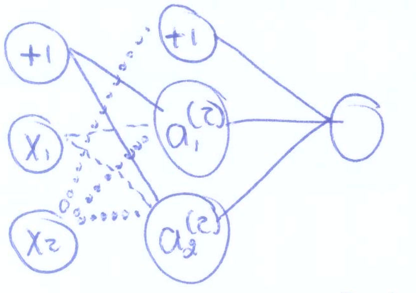
We need to pick some initial value for Θ.

Initializing Θ with zeros worked OK for logistic regression but is doesn't for (NNs).

The reason is that the network will be redundant.

$a_1^{(2)} = a_2^{(2)}$ and $\delta_1^{(2)} = \delta_2^{(2)}$.

After each update, parameters corresponding to inputs going into each of two hidden units are identical



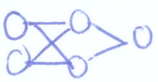
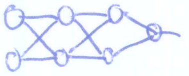
$\boxed{a_1^{(2)} = a_2^{(2)}}$ if $\Theta = 0$ for all elements in the beggining.

This is known as the problem of symmetric weights. For breaking the symmetry, we must iniatialize $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$ (, $\underbrace{-\varepsilon \leq \Theta_{ij}^{(l)} \leq \varepsilon}_{\text{this means}}$)

(This is not ε of before.

x

Wrapping up

The first thing we need to do when training a (NN), is to select a (NN) architecture (connectivity pattern)  or  or (...)

How to choose this?

① input units: dimensionality of the problem $x^{(i)}$.

② output units: number of classes.

③ for the internal (hidden layers), a reasonable default is $\boxed{\text{One}}$ hidden layer. But if we are using more than one hidden layer, it is interesting to have the same number of units in all layers (the more hidden layers, the better, normally).

Note: the more hidden layers, the better at the cost of much more computation.
mbR of units in hidden layer normally is comparable to that in the input layer.

Training a neural network

① Initialize weights randomly.

② Implement forward propagation to obtain $f_\Theta(x^{(i)})$ $\forall$ $x^{(i)}$.

③ Compute cost function $J(\Theta)$.

④ Backpropagation to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

Batch

Example

```
for i=1 to (m) {  mbR examples   // x^(1), y^(1); ...; x^(m), y^(m).
    Perform forward prop. and backprop using x^(i), y^(i).
    (Get activations a^(l) and delta terms δ^(l) for l=2, ..., L).
    Δ^(l) ← Δ^(l) + δ^(l+1) (a^(l))^T.   // accumulators having, in the end, the partial derivatives
                                          // ∂/∂Θ_ij^(l) J(Θ)
}
compute ∂/∂Θ_jk^(l) J(Θ)
```

⑤ Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed in ④ with a numerical estimate of gradient of $J(\Theta)$.

⑥ Disable gradient checking.

⑦ Use Gradient Descent or other opt method with backprop to minimize $J(\Theta)$ as a function of params Θ.

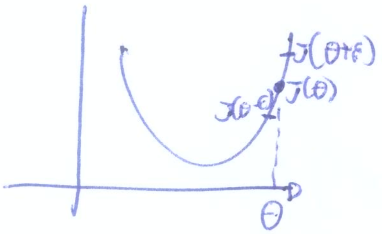
As we have $J(\Theta)$ as a non-convex function, we may ~~have~~ converge to local minima.

Backprop → calculates the derivatives (direction of a step) the gradient.

Gradient Descent → take little baby steps downhill.

Last notes on Gradient Checking

- Always check the first results of Gradient Descent using gradient checking.



- Gradient checking is too slow. Do not use it for learning, just to check if it works in the first iteration.