COMP7606/DASC7606-1B
Deep Learning

Linear Models (in Machine Learning)

Dr Bethany Chan
Professor Francis Chin

2021

Outline

Fundamental Machine Learning Techniques

- Introduction to Machine Learning
- Linear Classification
- Linear Regression
- Logistic Regression
- Binary vs. Multi-classification

Outline

Fundamental Machine Learning Techniques

- **Introduction to Machine Learning**
- Linear Classification
- Linear Regression
- Logistic Regression
- Binary vs. Multi-classification

Nature of machine learning (ML) algorithms

Algorithms that **learn** from **experience** (past data) to make **accurate predictions**

learning $\Rightarrow$ the algorithm **improves** with **more experience** to make **more accurate predictions**

Why would we be interested in developing algorithms that improve their performance over time with experience?

- Many algorithms that solve real world problems, which get the job done well, don't improve over time.
- E.g. algorithms ranging from banking and e-commerce
  to navigation systems in our cars and
  landing a spacecraft on the moon
- Why do we need machine learning?
- **Answer: for certain tasks it is easier to develop an algorithm that learns/improves its performance with experience than to develop an algorithm manually.**

When should you use ML?

- Pattern is expected to exist in the data
- Have lots of data to learn from

How much data is enough?

Sample complexity

sample size (amount of data)

required to learn a family of concepts

How did we learn in school?

Asked **Questions**
given **Answers**

Questions & Answers
→ Practice exercises (**training data**)
→ Mock exams (**validation**)
→ Final exams (**testing**)

Supervised Learning

Learning Scenarios

Supervised learning

Given **data**, predict **labels**

Unsupervised learning

Given **data**, learn about that **data**

Reinforcement learning

Given **data**, choose **action** to maximize expected **long-term reward**

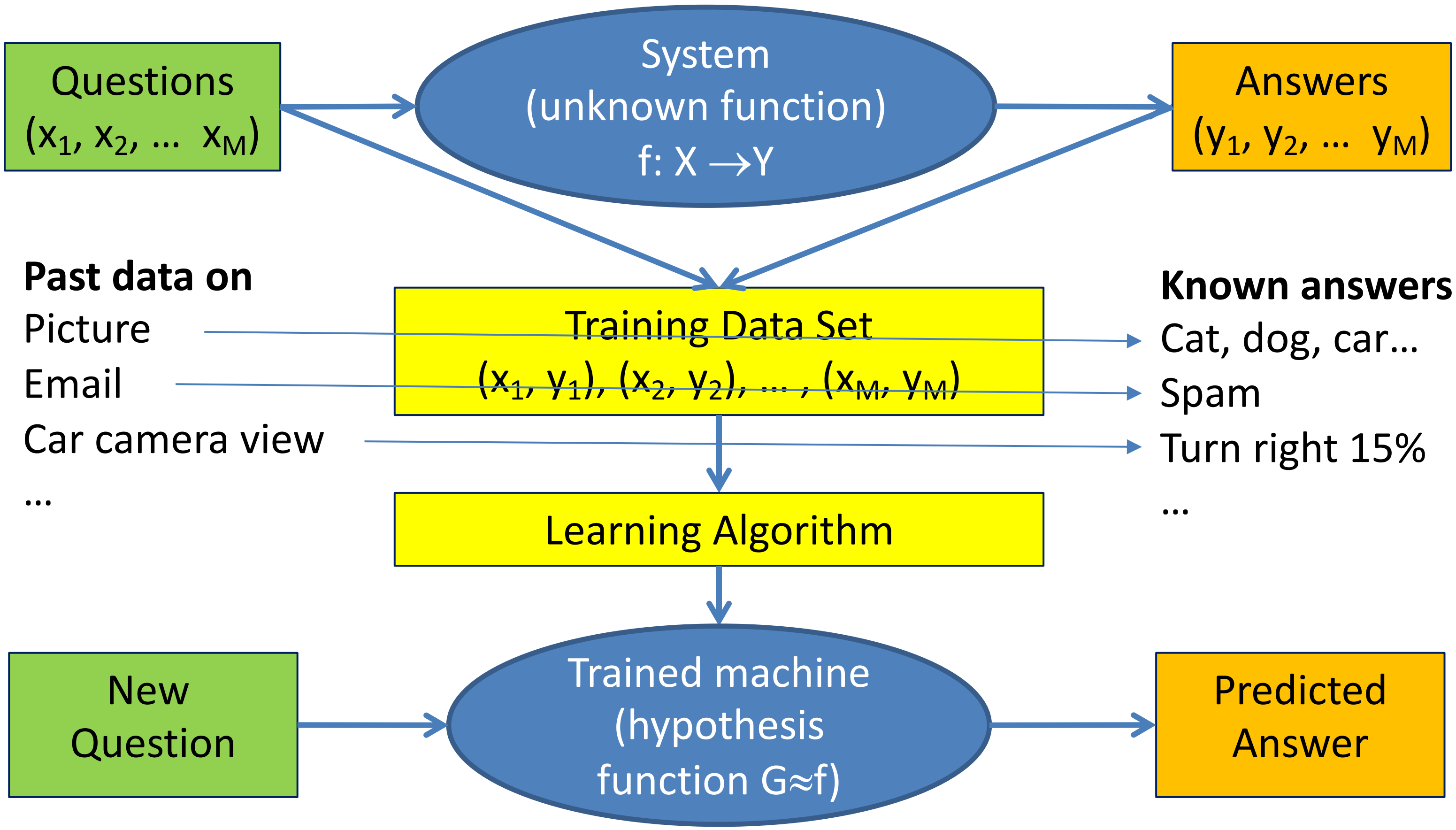
Supervised Learning

Learning scenario where…
for each example, told what correct label is
Given **data**, predict **labels**

Classification problems
(discrete labels)
e.g. determine whether e-mail is spam or not

Regression problems
(continuous labels)
e.g. how much you should turn the steering wheel

Machine Learning Model (supervised learning)



Outline

Fundamental Machine Learning Techniques

- Introduction to Machine Learning
- **Linear Classification**
- Linear Regression
- Logistic Regression

Classification

Problem:

Given a set of labeled (*classified*) data points (**x**,y), learn a function f: X $\rightarrow$Y in order to predict the label (*class*) of new unseen data points **x**′

- **Binary** Classification:
  Y has only two values, e.g. Y = {-1, +1}
- **Multi-class** Classification:
  Y has more than two values

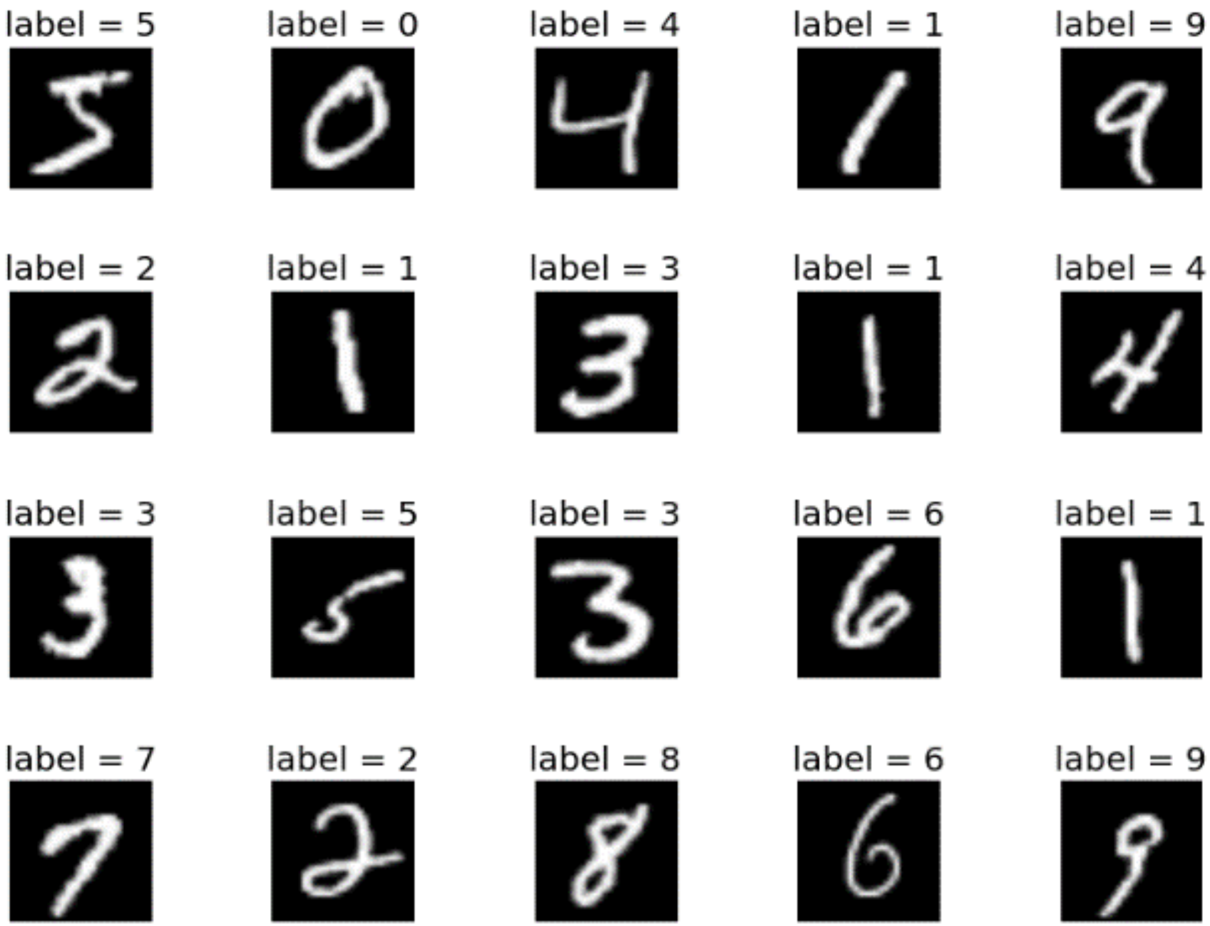
Problem: credit approval

Applicant information: $\boldsymbol{x}$ = $(x_1,\ldots, x_N)$

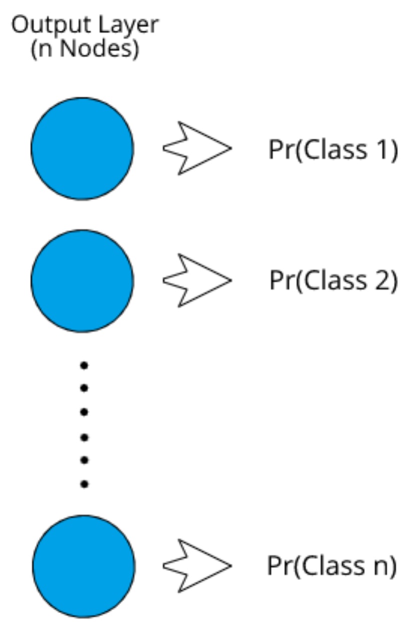
x_1	salary	150,000
x_2	current debt	75,000
x_3	age	28 years old
x_4	years in current job	3
…	…	…

Approve or *deny* credit? $\Rightarrow$ binary classification

Problem: handwritten digits

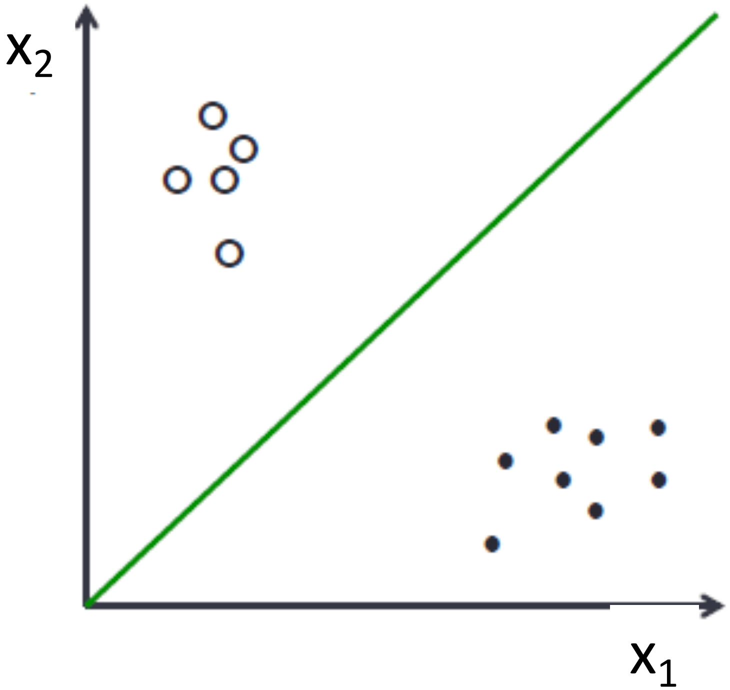


labels Y = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} $\Rightarrow$ *multi-class*


Output Layer
(n Nodes)
Pr(Class 1)
Pr(Class 2)
Pr(Class n)

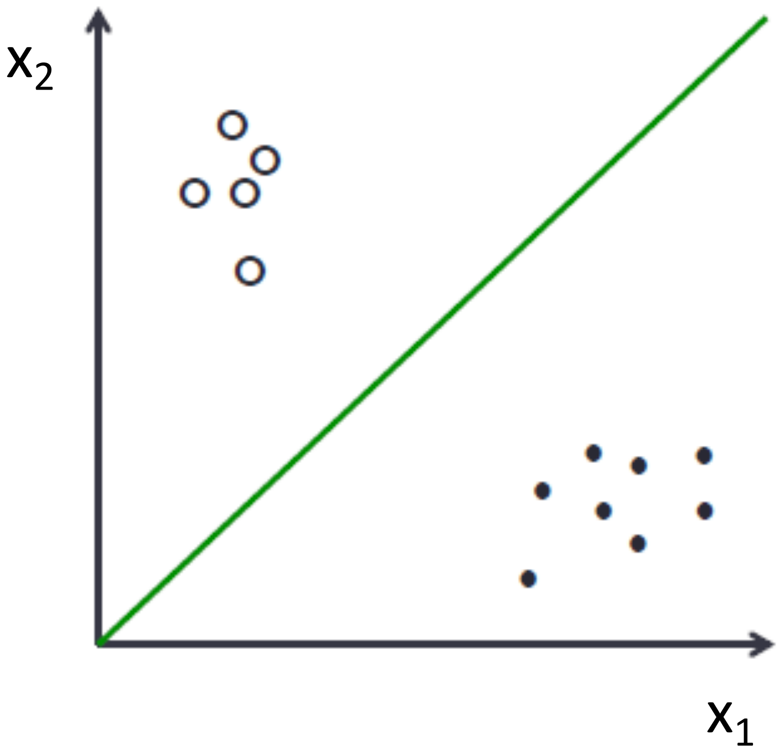
Linear Classification

A **linear classifier** achieves this by making a classification decision based on the value of a linear combination of the characteristics.



The data points can be separated by a hyperplane (here a line) that helps to correctly classify all points.

Linear Classification



Line (decision boundary)

Equation of the line:
$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots = 0$

If we introduce $x_0 = 1$:
$\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots = 0$

Then we can use $\boldsymbol{\theta x} = 0$ as shorthand for the equation

Linear Classification

$\boldsymbol{\theta x} = 0 \; (dot\ product)$

$i.e., x_1 - x_2 = 0$

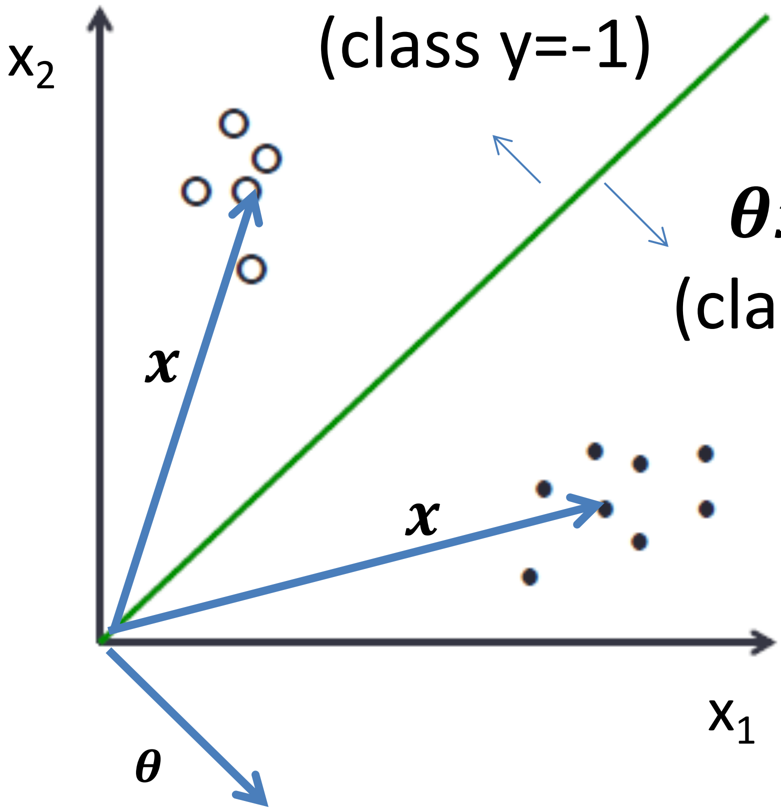
(decision boundary)

$\boldsymbol{\theta x} < 0$

(class y=-1)

$\boldsymbol{\theta x} > 0$

(class y=+1)



Problem of classification becomes problem of finding the line (or decision boundary), or finding θ's

Problem: credit approval

Applicant information: $\boldsymbol{x}$ = $(x_1,\ldots, x_N)$

x_1	salary	150,000
x_2	current debt	75,000
x_3	age	28 years old
x_4	years in current job	3
…	…	…

Approve credit if $\sum_{j=1}^{N} \theta_j x_j \geq \textit{threshold}$

Deny credit if $\sum_{j=1}^{N} \theta_j x_j < \textit{threshold}$

Problem: credit approval

Applicant information: $\boldsymbol{x} = (x_1, \ldots, x_N)$

x_1	salary	150[illegible]
x_2	current debt	
x_3	age	
x_4	years in current [illegible]	
…	…	…

θs TO BE LEARNED

Approve credit ~~if $\sum_{j=1}^{N} \theta_j x_j \geq$ *threshold*~~ if $\boldsymbol{\theta x} \geq 0$

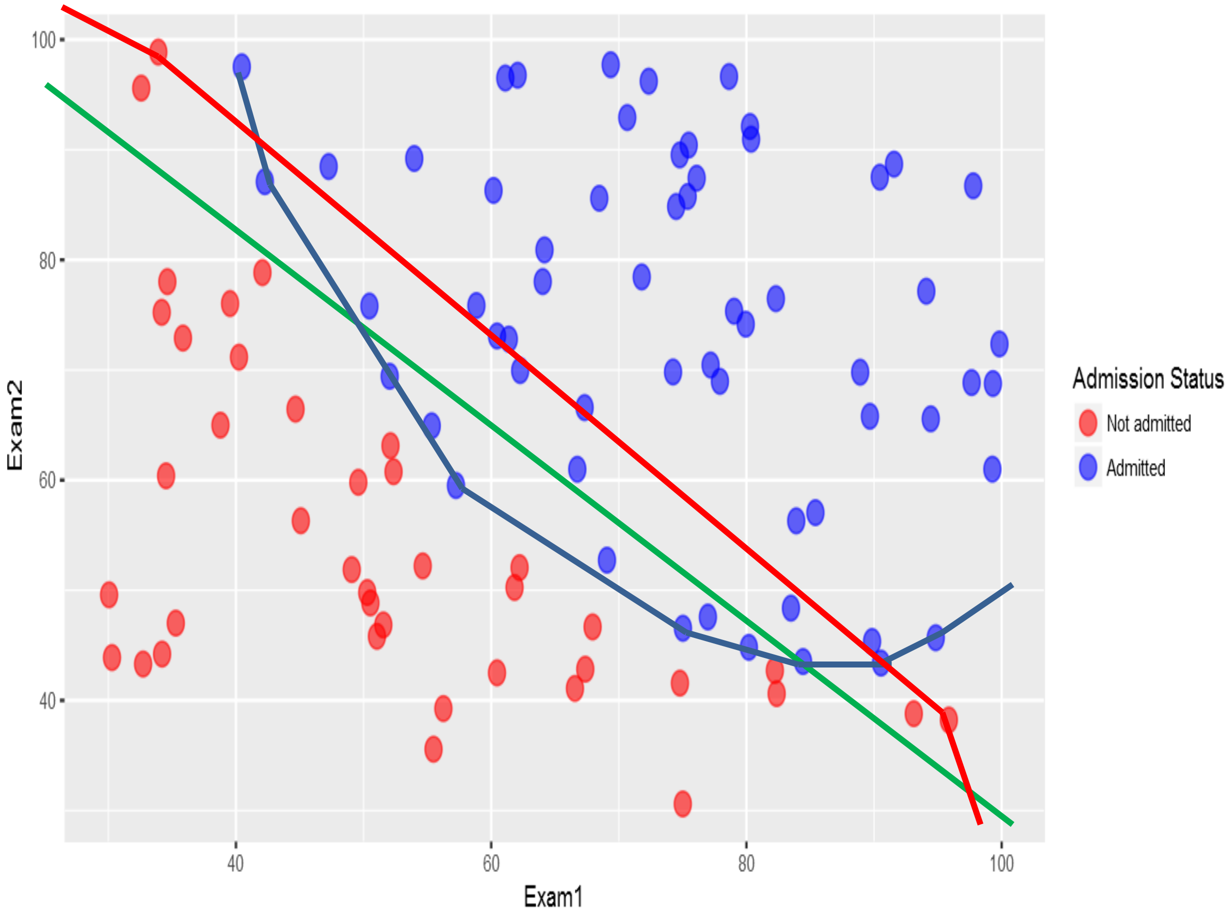
Deny credit ~~if $\sum_{j=1}^{N} \theta_j x_j <$ *threshold*~~ if $\boldsymbol{\theta x} < 0$

[θ_0= -*threshold* and $x_0 = 1$]

How to find the Linear Classifier?

Not possible!

How to find the best.



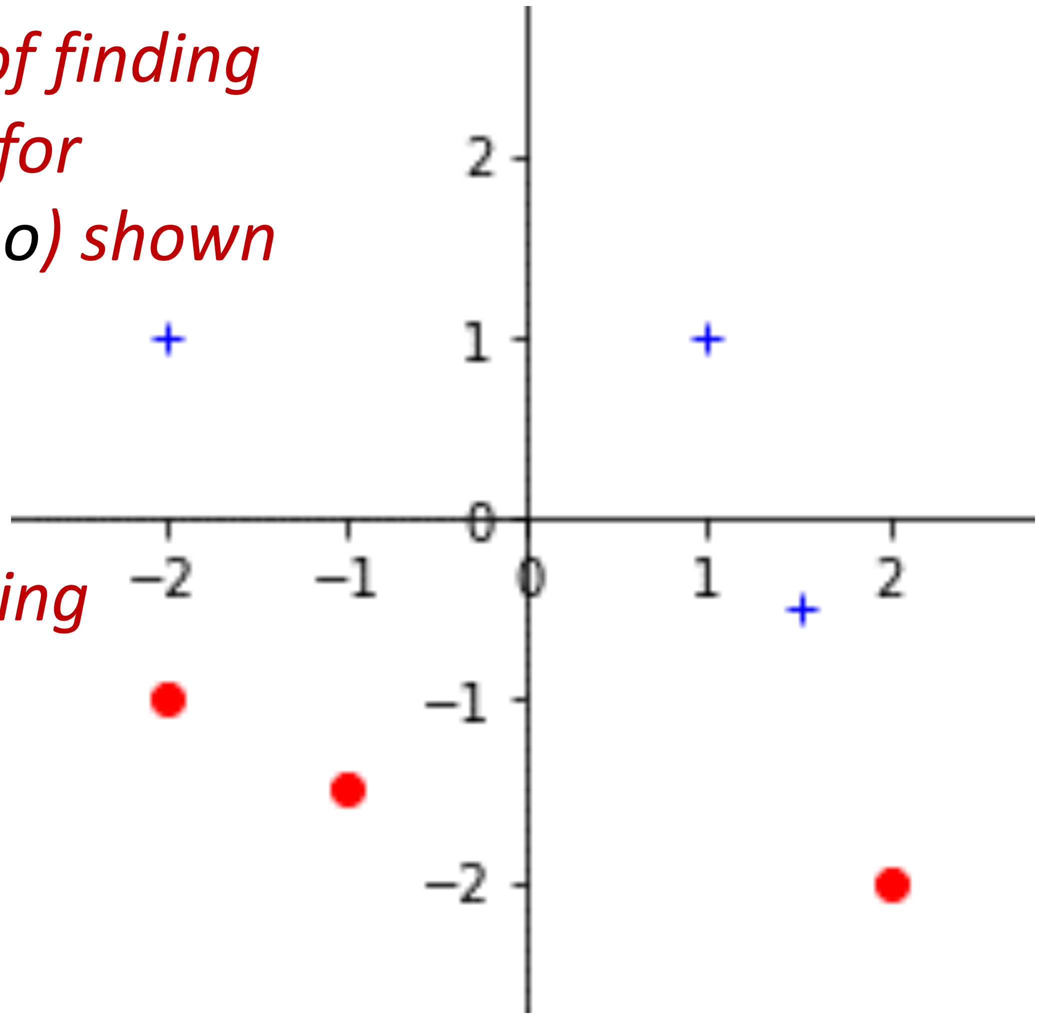
Shortcomings of Classifier Algorithms

- Complicated
- Problem with outliners
- updates (new or obsolete data)
- Dynamic situation
  (classification criteria might change with time)

An Algorithm to find Linear Classifier

Consider the problem of finding the decision boundary for the 6 points (3 + and 3 o) shown

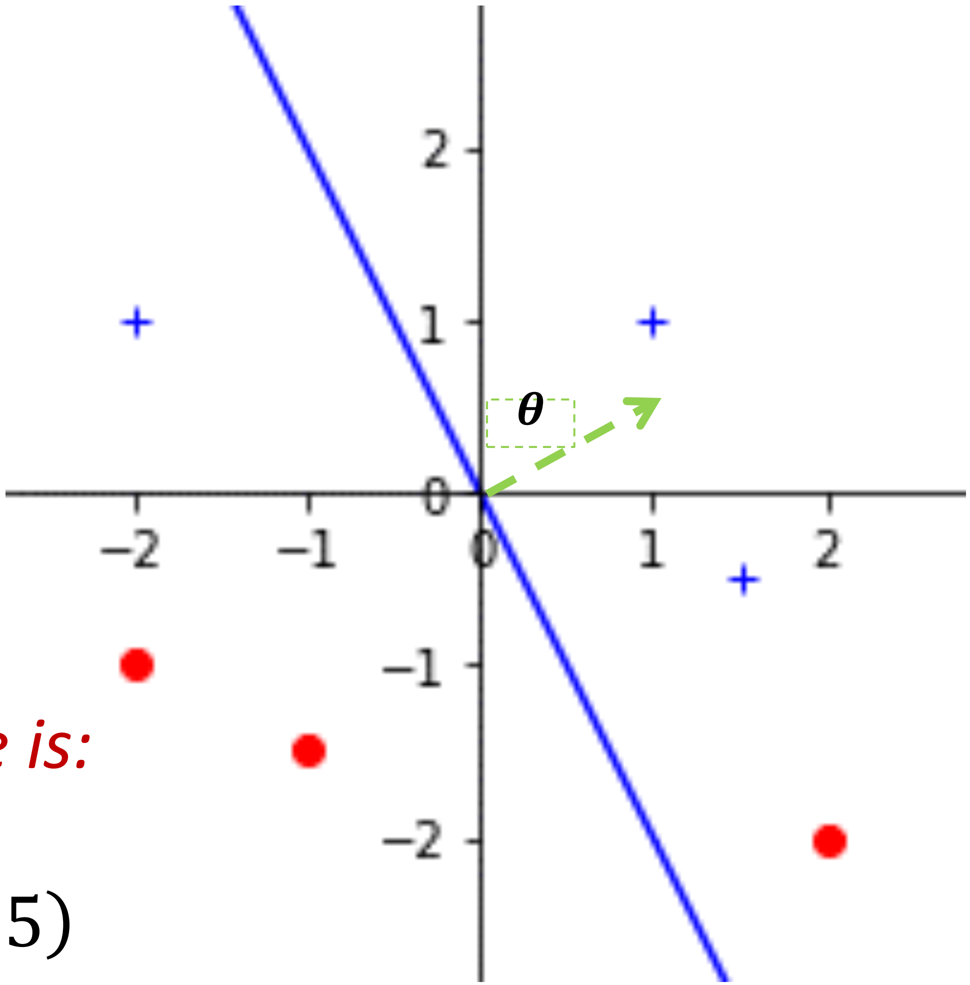
- *Online*
- *Iterative*
- *close to Deep Learning*



Start with
a random line,
say, the one shown

Equation of this line

$$x_1 + 0.5x_2 = 0$$



Equation of this line is:

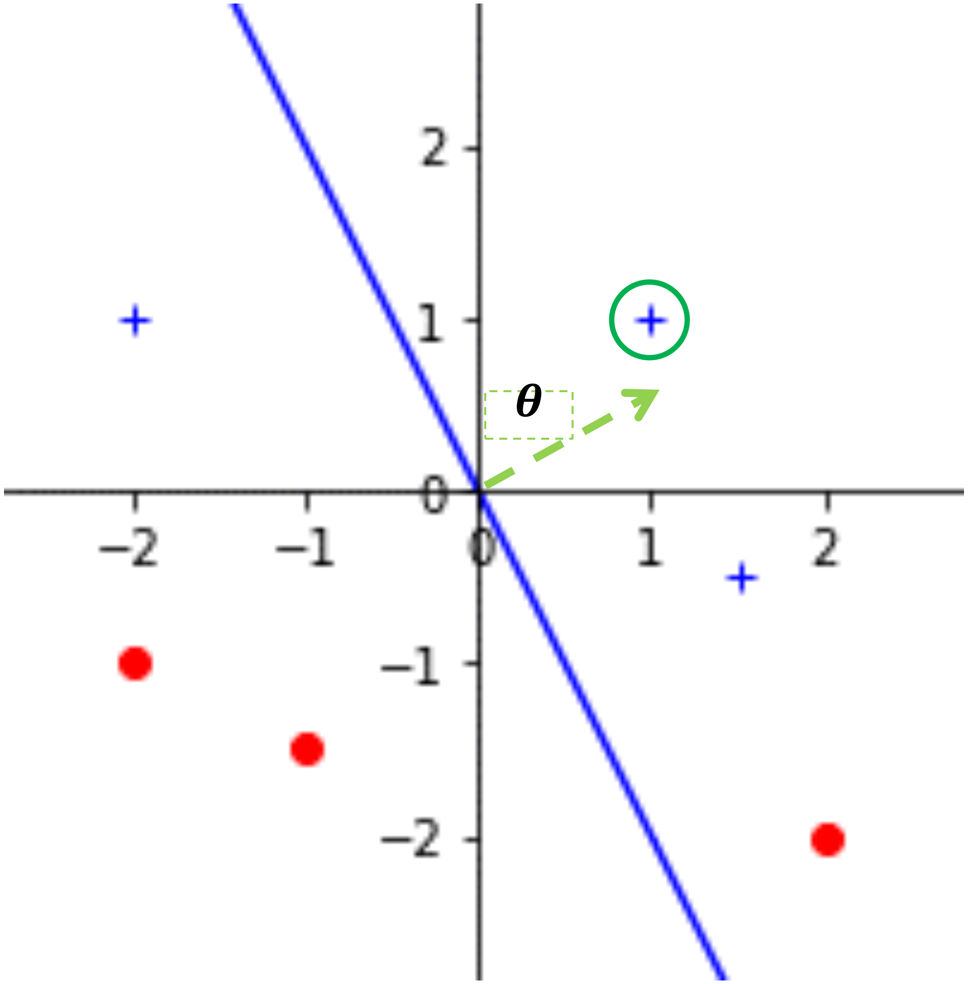
$$\boldsymbol{\theta x} = 0$$

where $\boldsymbol{\theta} = (0, 1, 0.5)$

and $x_0 = 1$

Consider the points one by one, say, the one circled:

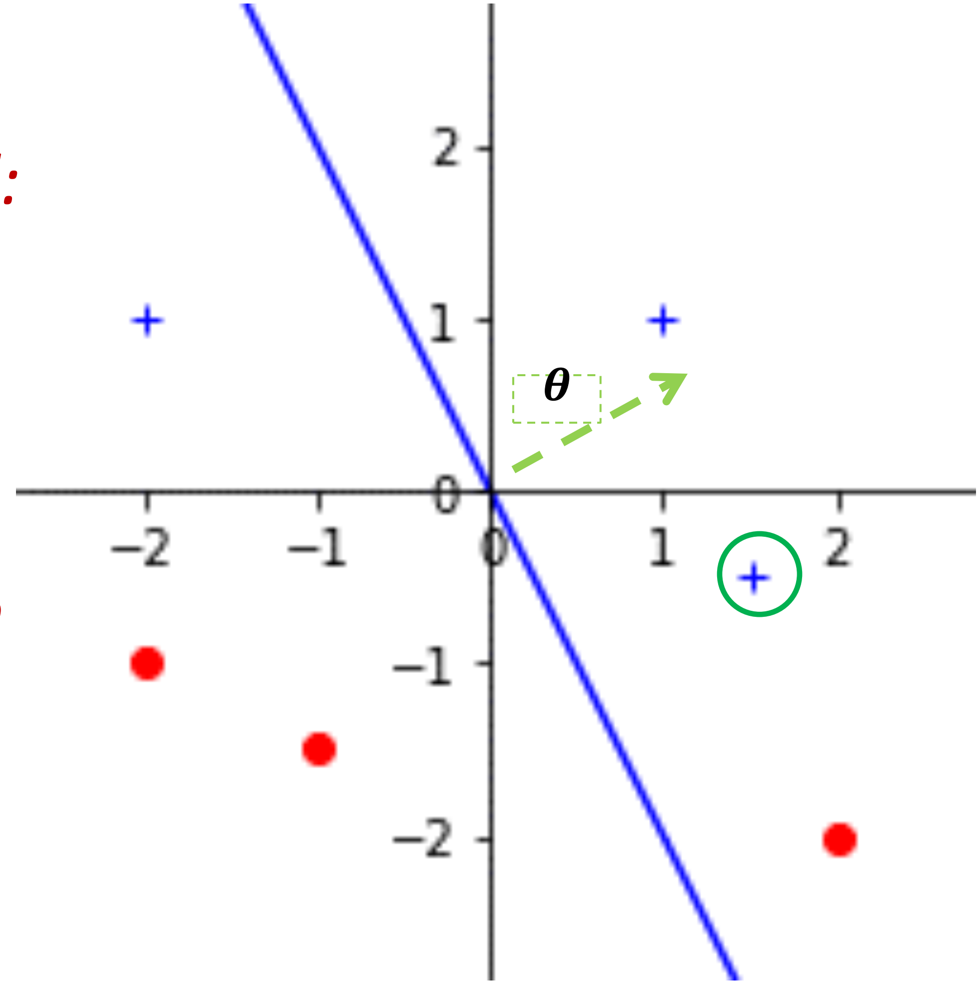
(x_1, x_2) =(1,1)



Correct classification since $\boldsymbol{\theta x} > 0$

Consider next circled:
(x_1, x_2) = (1.5, -0.5)

Correct classification since $\boldsymbol{\theta x} > 0$

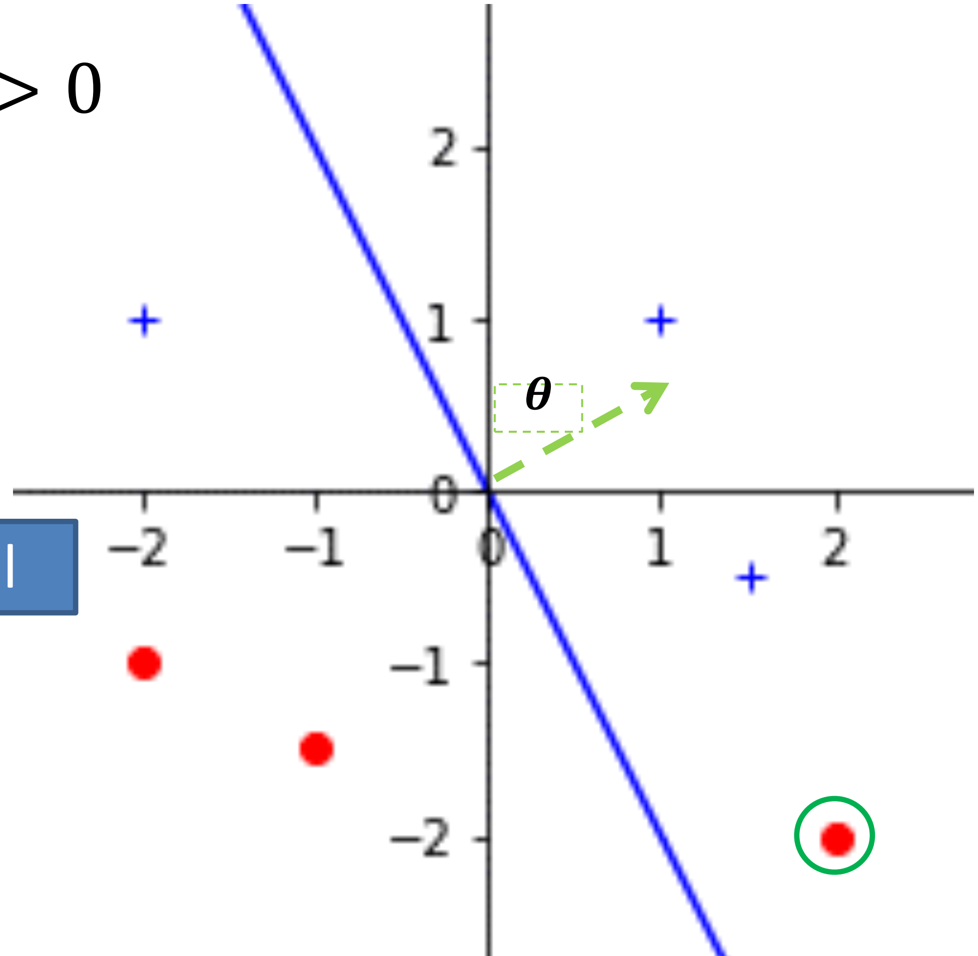


Consider circled: (x_1, x_2) = (2,-2)

Misclassified! *since* $\boldsymbol{\theta x} > 0$

Need to adjust the line.

But how?



Correct label

Use formula:

$$\theta \leftarrow \theta + \alpha y \boldsymbol{x}$$

where α *is a parameter.*

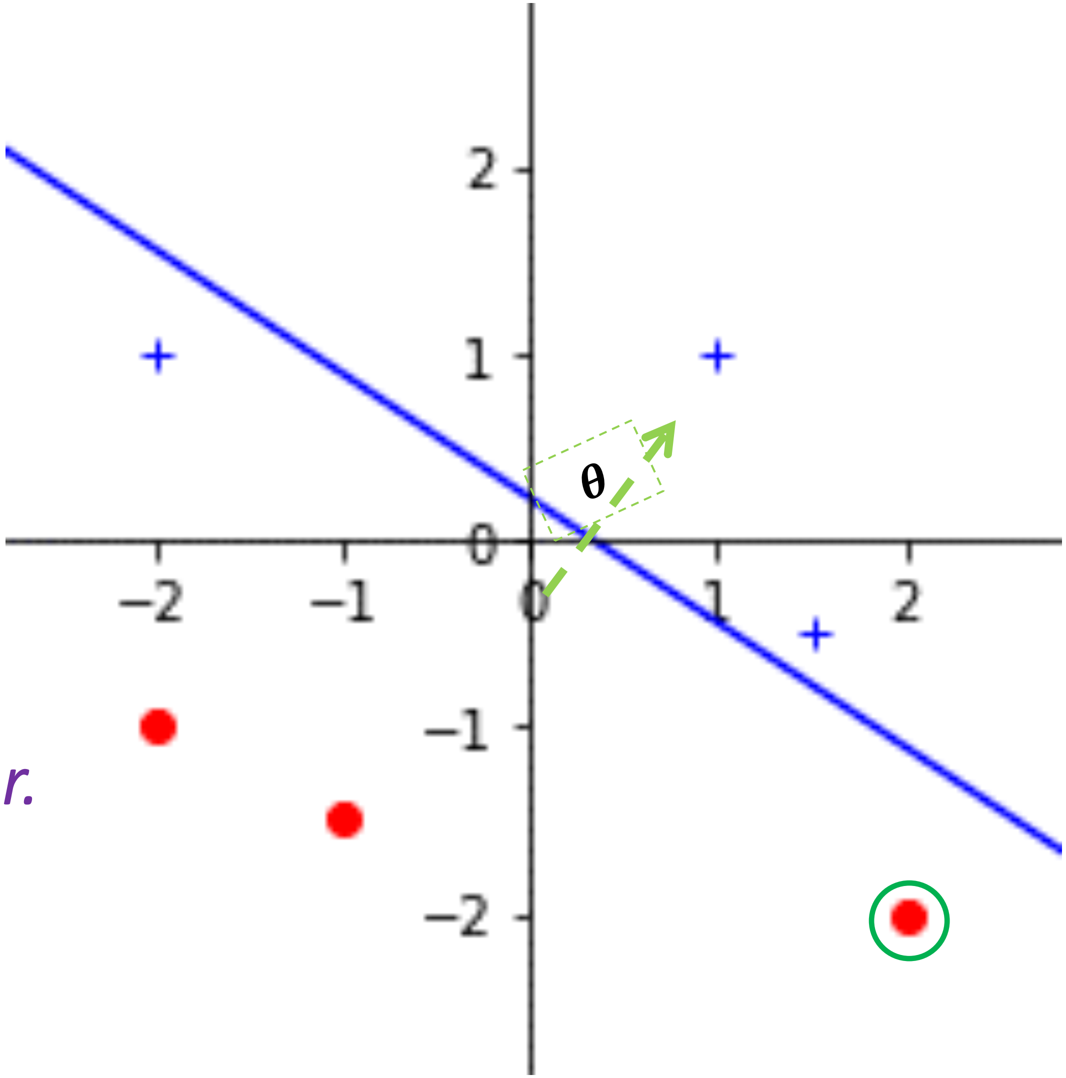
Let's set α*=0.2 here*

Rotate ... wise

"Learning rate"

Shift the classifier upward

$\boldsymbol{\theta}$ ← (0, 1, 0.5) + 0.2 * -1 * (1, 2, -2) = (-0.2, 0.6, 0.9)



Use formula:

$$\theta \leftarrow \theta + \alpha y \boldsymbol{x}$$

where α *is a parameter.*

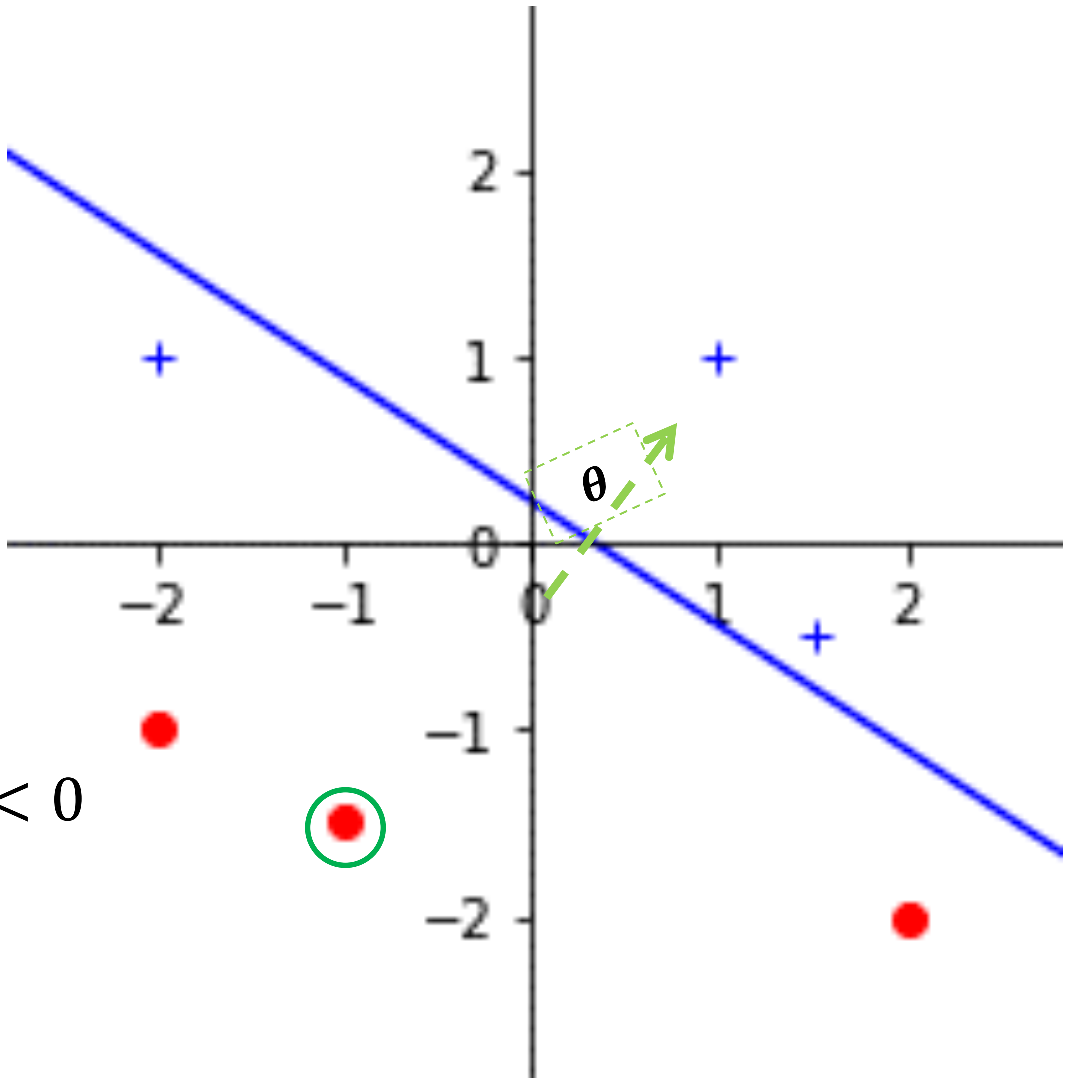
Let's set α *= 0.2 here*

$\boldsymbol{\theta}$ ← (0, 1, 0.5) + 0.2 * -1 * (1, 2, -2) = (-0.2, 0.6, 0.9)

$$0.6x_1 + 0.9x_2 = 0.2$$

Consider next circled:

(x_1, x_2) = (-1, -1.5)

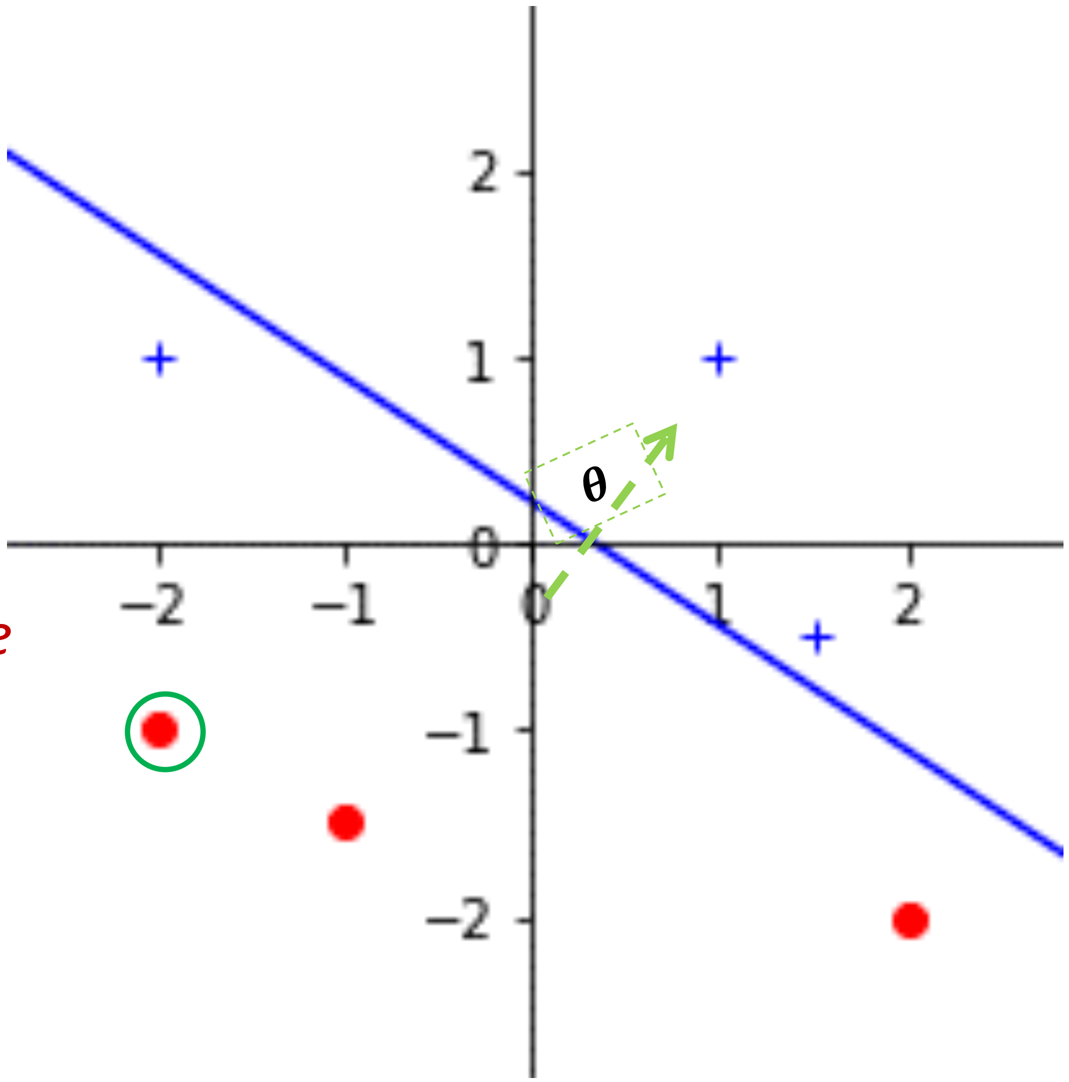


Correct classification since

$\boldsymbol{\theta x} < 0$

$- 0.2 + 0.6\ (-1) + 0.9\ (-1.5) < 0$

Consider next circled:
(x_1, x_2) = (-2, -1)



Correct classification since
$\boldsymbol{\theta x} < 0$
-0.2 + 0.6(-2) + 0.9(-1) < 0

Consider circled:

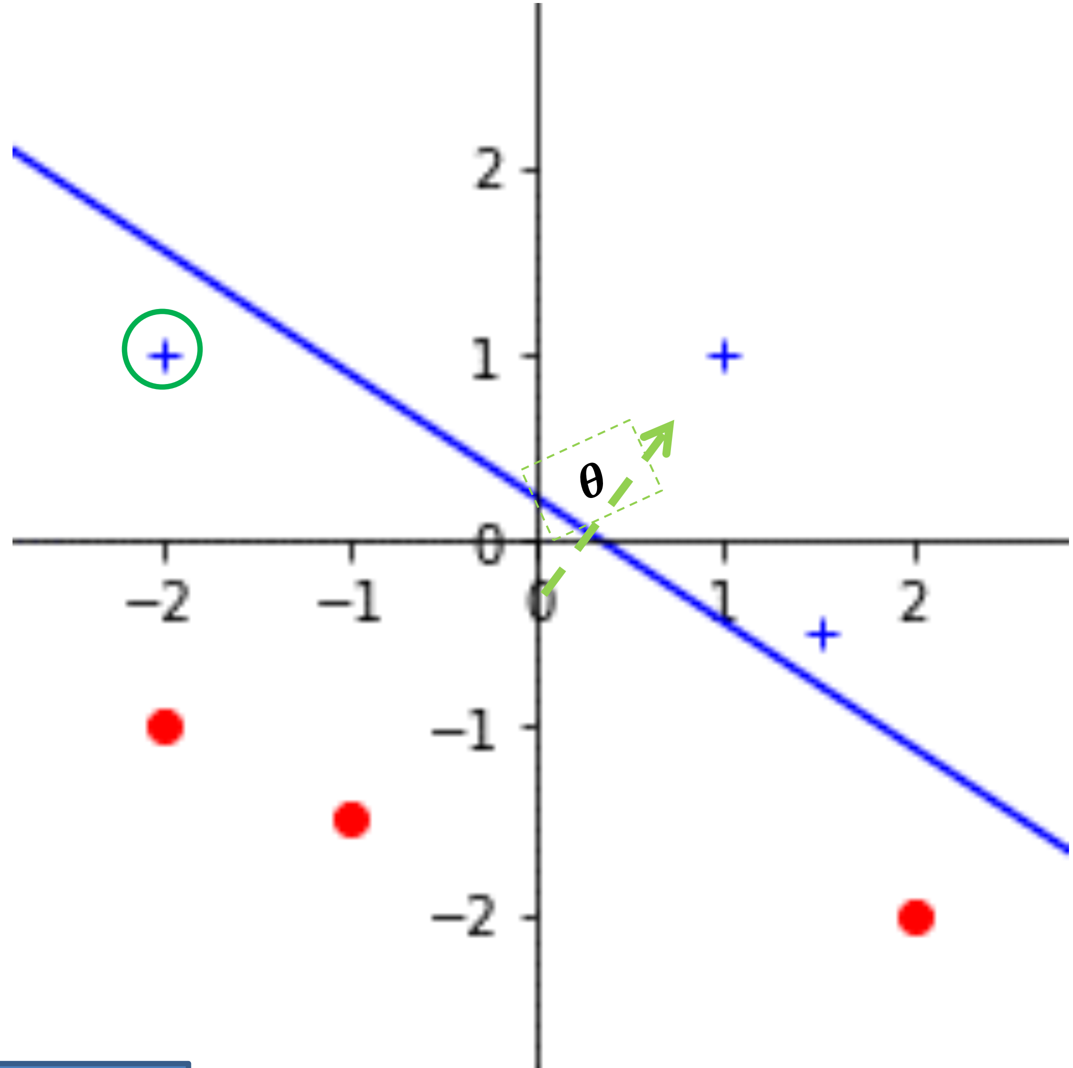
(x_1, x_2) = (-2, 1)

Misclassified!

$$\boldsymbol{\theta x} < 0$$

-0.2 + 0.6(-2) + 0.9(1) < 0

Need to adjust the line.



Again use formula:

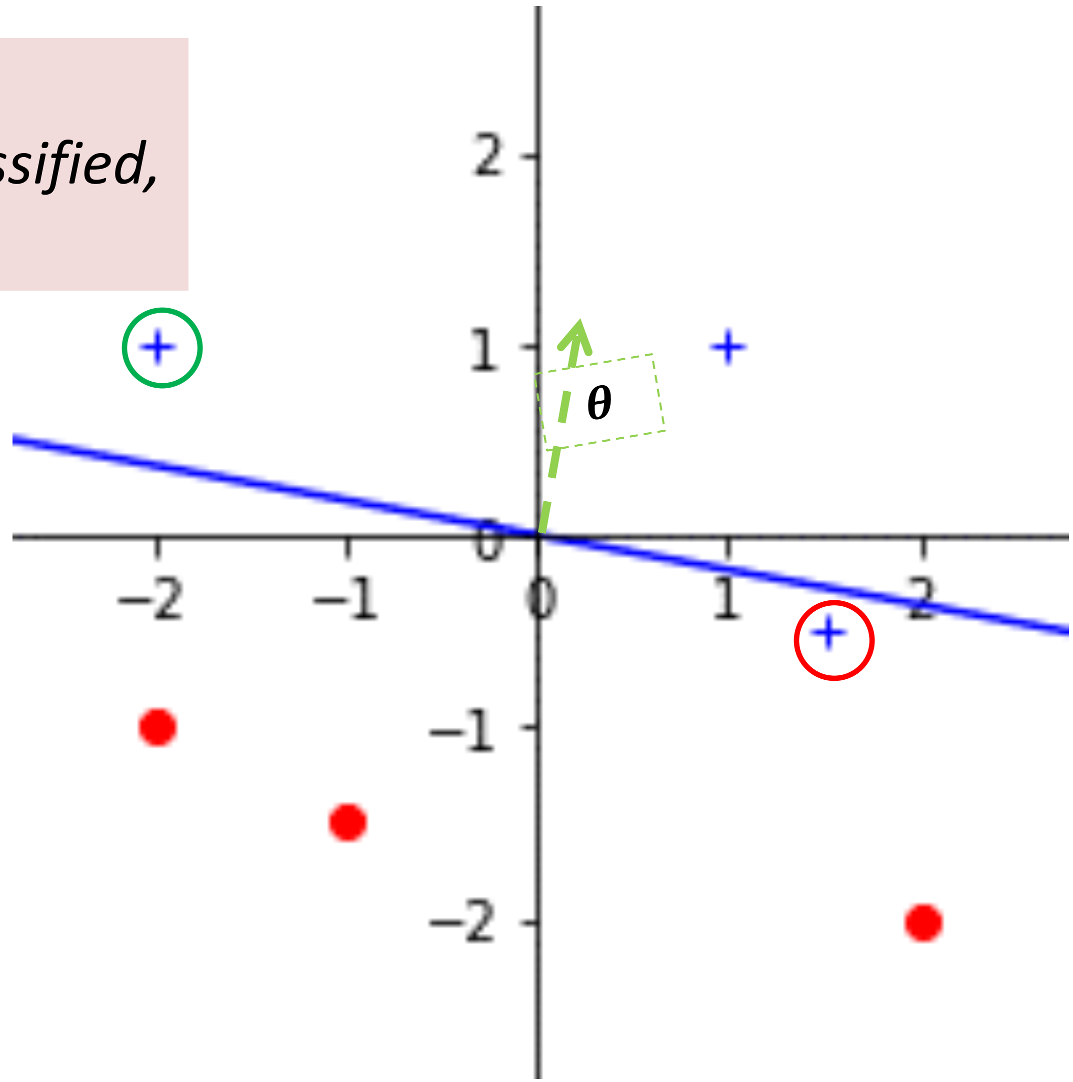
$$\theta \leftarrow \theta + \alpha y \boldsymbol{x}$$

with α = 0.2

Rotate θ counter-clockwise

$\boldsymbol{\theta}$ ← (-0.2, 0.6, 0.9) + 0.2 * 1 * (1, -2, 1) = (0, 0.2, 1.1)

Circled in red:
Originally correctly classified, now misclassified



Use formula:

$$\theta \leftarrow \theta + \alpha y \boldsymbol{x}$$

again with α *= 0.2*

$\boldsymbol{\theta} \leftarrow$ (-0.2, 0.6, 0.9) + 0.2 * 1 * (1, -2, 1) = (0, 0.2, 1.1)

Linear Classification

Given data set $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_M, y_M)$

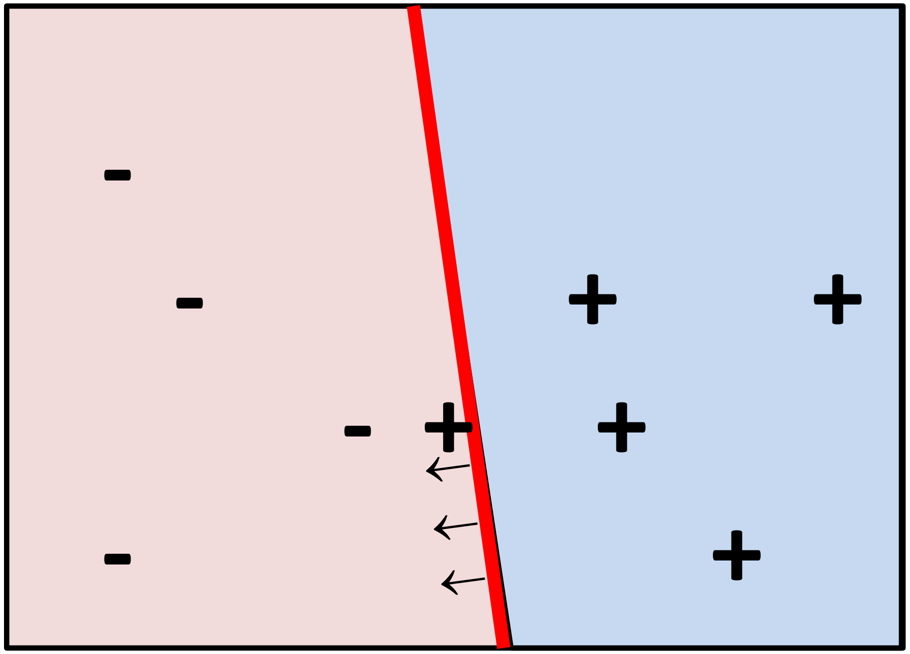
Make an initial guess for θ

Repeat for the data set

For each point $(\boldsymbol{x}_i, y_i)$:

check that the point is correctly classified

if misclassified, then **update θ**



Learning the weights θ,
try to improve with each data point

Not always possible to find a decision boundary that will separate the data

Takeaway Messages

- Parameters θ are to be learned
- Learned (by adjusting θ) by using the given labeled data iteratively and repeatedly (epoch)
- Adjustment amount depends on
  - Learning rate
  - Amount of error (loss function)
- Converge to the best solution independent of the initial guess of θ (only affect the efficiency)

Deep Learning takes similar approach

Outline

Fundamental Machine Learning Techniques

- Introduction to Machine Learning
- Linear Classification
- **Linear Regression**
- Logistic Regression
- Binary vs. Multi-classification

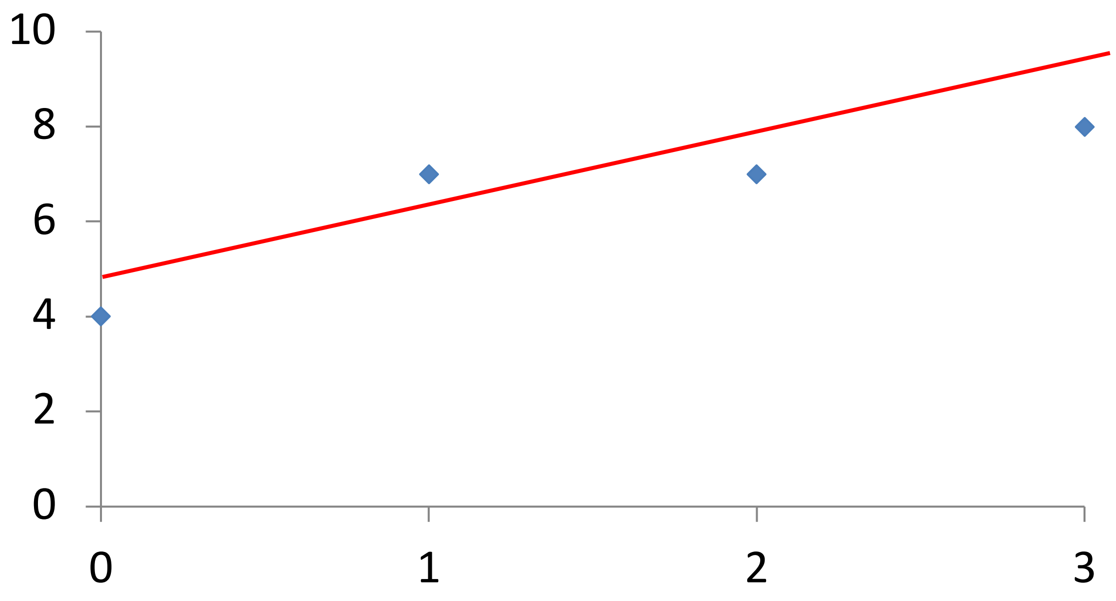
Regression

- Problem:
  Given a set of data points ($\mathbf{x}$,y),
  learn a function f: X $\rightarrow$Y
  in order to predict the label (*value*)
  of new unseen data points $\mathbf{x}'$

Linear Regression with One Variable

- Predict single output y from **single input** x
- Linear hypothesis h :

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



Another example

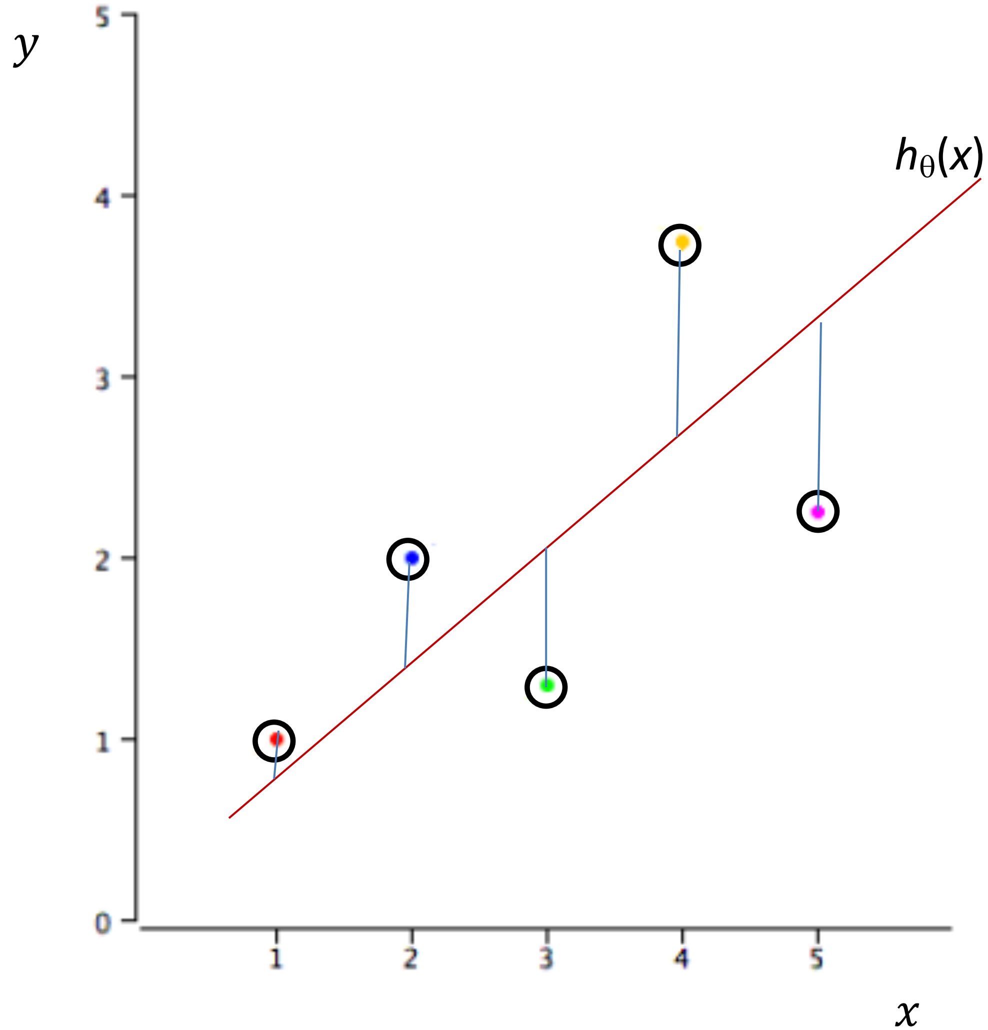
x	y
1.00	1.00
2.00	2.00
3.00	1.30
4.00	3.75
5.00	2.25

Norm2 error (L^2)

$= \sqrt{\sum_{i=1}^{n} (h_\theta(x_i) - yi)^{2}}$

Mean square error

$= \frac{1}{n} \sum_{i=1}^{n} (h_\theta(x_i) - yi)^2$



Loss and Cost function

- In ML, we want to **minimize** the error (defined by the loss function) for each training example during the learning process.
- A loss function maps decisions to their associated costs, like gradient descent (to be learned later).
- A loss function is for a single training example (sometimes called **error function)**. A cost function is the **average loss** over the entire training dataset, which optimization strategies aim to minimize.

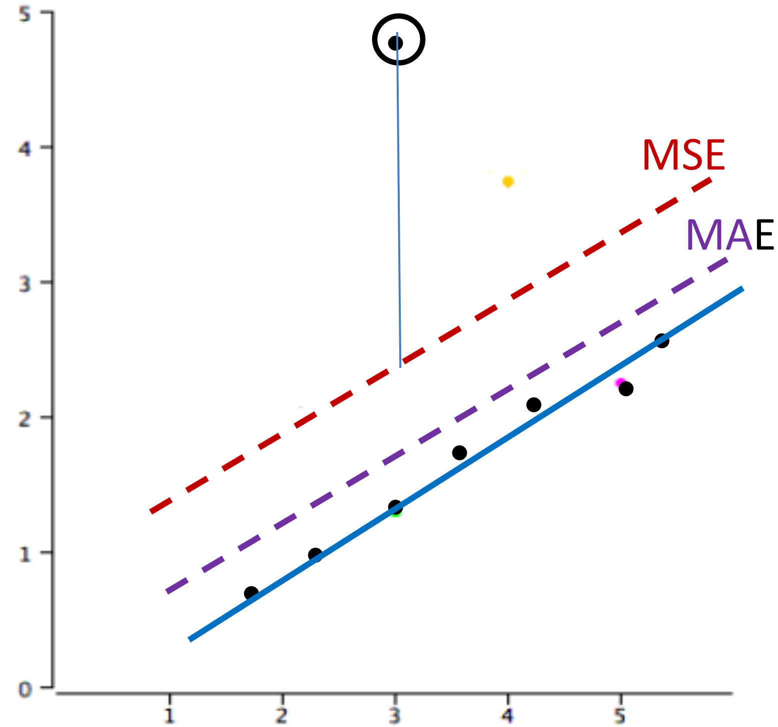
Loss Function (one variable)

- Machines learn by means of a **loss function**.
- **Loss functions** (denoted by J) evaluate how well your algorithm models your dataset.
- A good cost function which can **penalize** a model effectively while it is training on a dataset.
- If predictions are off, the **loss function**/**cost** is high.
  - this huge value will be used to change the parameters θ
  - the weights will be changed more than usual.
- If predictions are good, the **loss function**/**cost** be low.
  - the parameters θ won't change much.

- Different loss functions for different problems, broadly categorized into 2 types:
  Classification (-1, 1) and Regression Loss.

Regression Loss Functions

- Squared Error Loss:
  $(h(x) - y)^2$
  Cost = MSE (mean sq error)
- Absolute Error Loss:
  $| h(x) - y |$
  Cost function (Mean MAE)
- MAE is more robust to outlier



- Huber Loss: $\begin{cases} \frac{1}{2}(h_\theta(x) - y)^2 & if\ |h_\theta(x) - y| \leq \delta \\ \delta\, |h_\theta(x) - y| - \frac{1}{2}\delta^2 & otherwise \end{cases}$

  combine MSE and MAE, quadratic for small error, and is linear otherwise.

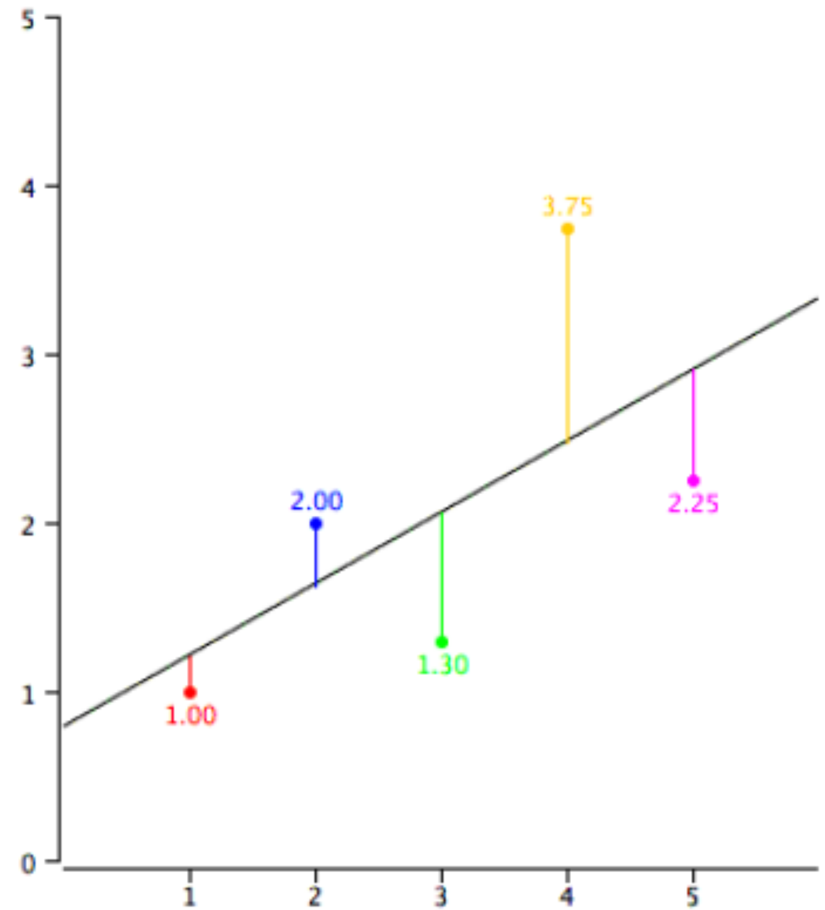
Look at cost $\frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)^2$ $(norm\ 2)$

$where \quad h_\theta(x_i) = \theta_0 + \theta_1 x_i = 0.785 + 0.425 x_i$

x	y	$h(x)$	$h(x) - y$	$(h(x) - y)^2$
1.00	1.00	1.210	0.210	0.044
2.00	2.00	1.635	-0.365	0.133
3.00	1.30	2.060	0.760	0.578
4.00	3.75	2.485	-1.265	1.600
5.00	2.25	2.910	0.660	0.436

$$\theta_0 = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n(\sum x^2) - (\sum x)^2}$$

$$\theta_1 = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$



Iterative way to learn θ

Make an initial guess for θ

Calculate the error $J(\theta)$

Repeat until error is small enough:

    make a better guess for θ

    calculate the error $J(\theta)$

Return θ

Iterative way to learn θ

Make an initial guess for θ

Calculate the error $J(\theta)$

Repeat until error is small enough:

  make a better guess for θ

  calculate the error $J(\theta)$

Return θ

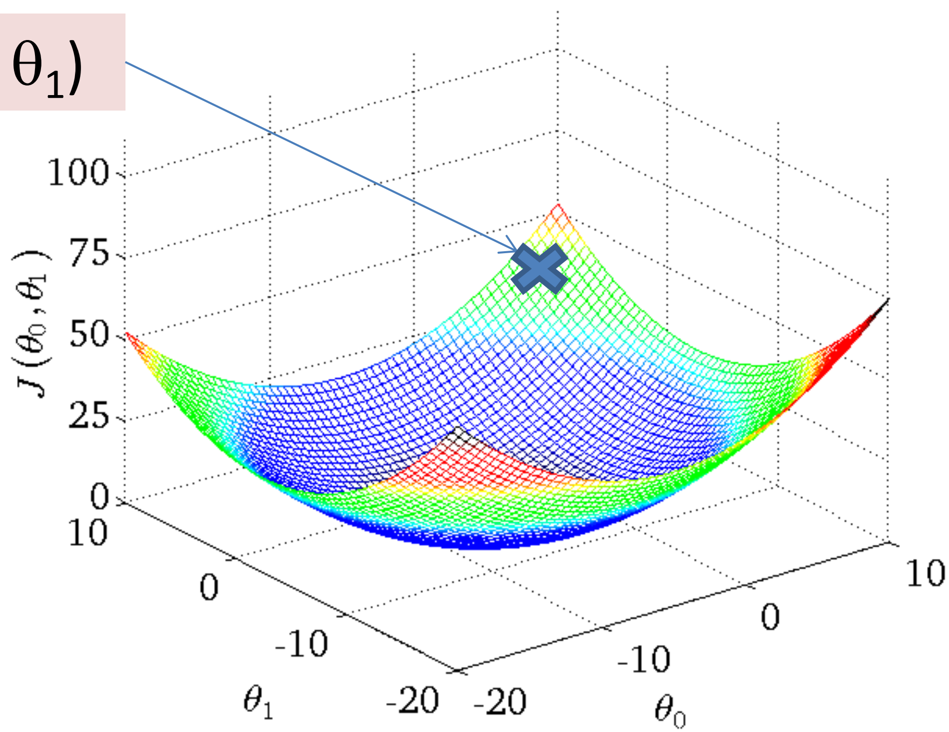
Technique: **Gradient Descent**

Gradient Descent (one variable)

Objective is to minimize cost function:

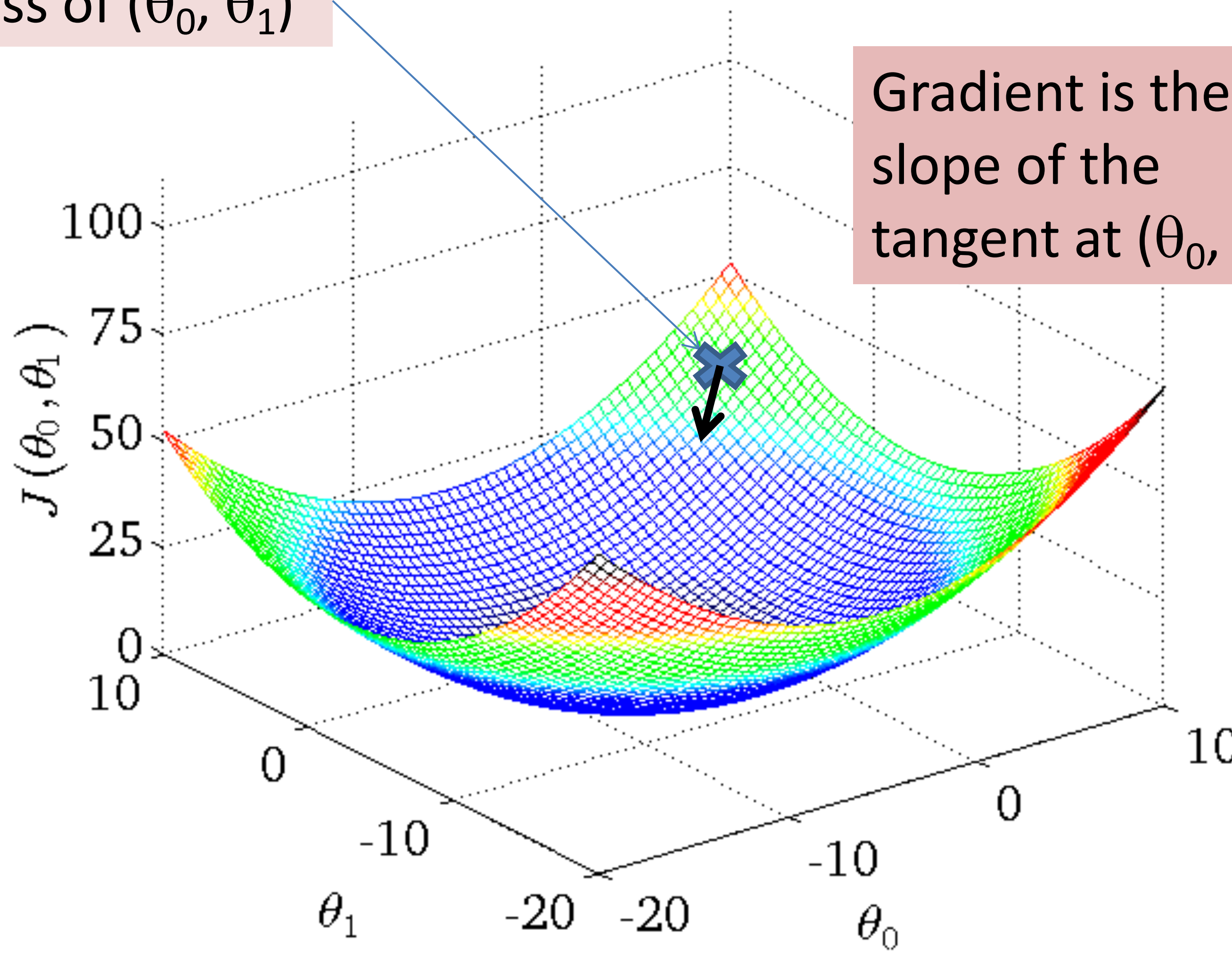
$$J(\theta_0, \theta_1) = \frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)^2$$

Random guess of (θ_0, θ_1)

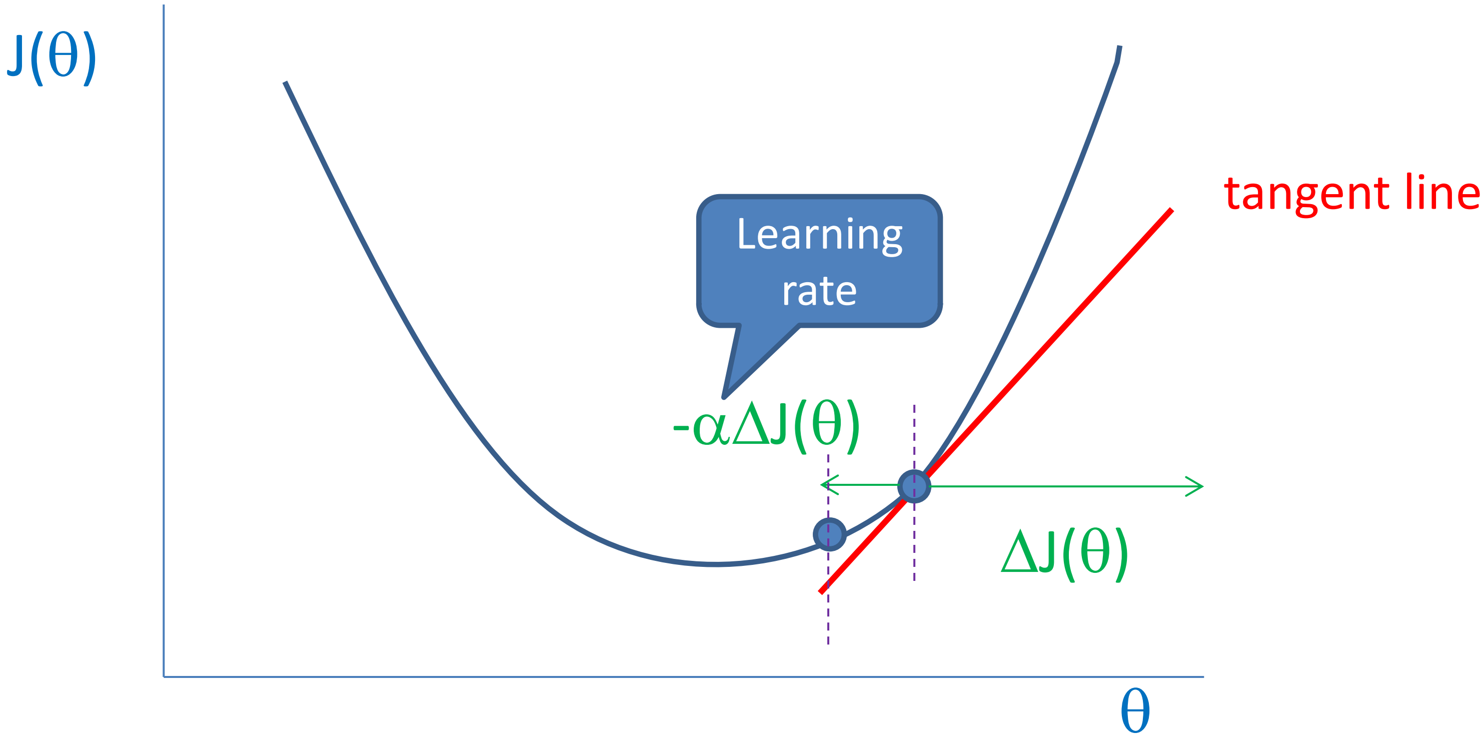


Random guess of (θ_0, θ_1)

Gradient is the slope of the tangent at (θ_0, θ_1)



Move in opposite direction of gradient (steepest descent)



Move in opposite direction of gradient (steepest descent)

Step size is determined by the gradient magnitude $\Delta J(\theta)$,

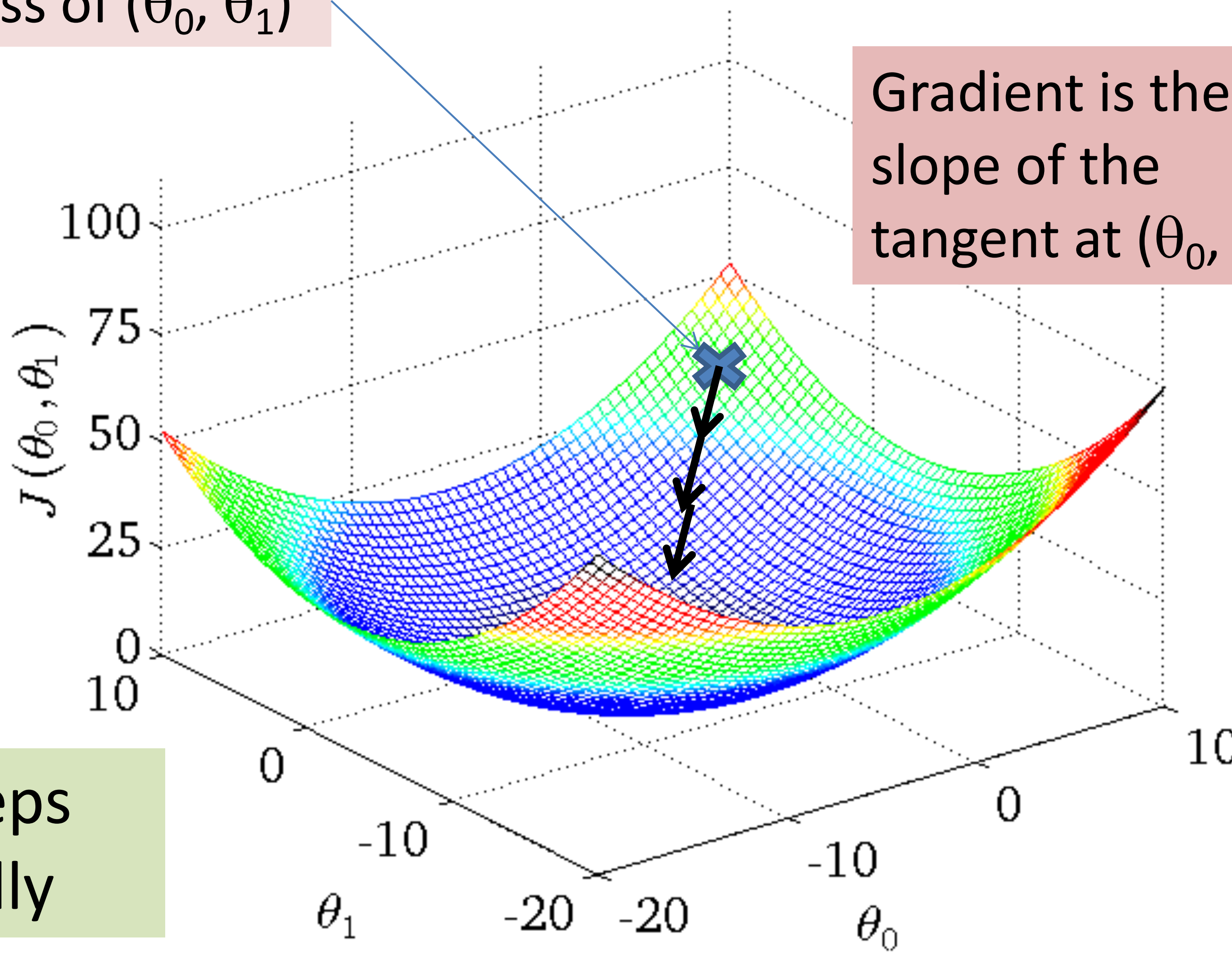
larger step size for steeper tangent line

Small step when close to the minimum, $\Delta J(\theta)$ is small

as the tangent line is almost horizontal.

Random guess of (θ_0, θ_1)

Gradient is the slope of the tangent at (θ_0, θ_1)



Make steps repeatedly

Move in opposite direction of gradient (steepest descent)

Gradient Descent (one variable)

Make steps of size α (**learning rate**) down the **cost function** *J* in the **direction with the steepest descent** (as determined by **slope of the tangent** at (θ_0, θ_1))

Repeat until error is small enough:

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

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

Cost Function for Gradient Descent

(one variable)

Mean-square error:

$$J(\theta_0, \theta_1) = \frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)^2$$

Half mean-square error :

$$J(\theta_0, \theta_1) = \frac{1}{2M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)^2$$

Taking derivatives (one variable)

$J(\theta_0, \theta_1) = \frac{1}{2M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)^2$ and $h_\theta(x_i) = \theta_0 + \theta_1 x_i$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)\, x_i$$

Repeat:

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

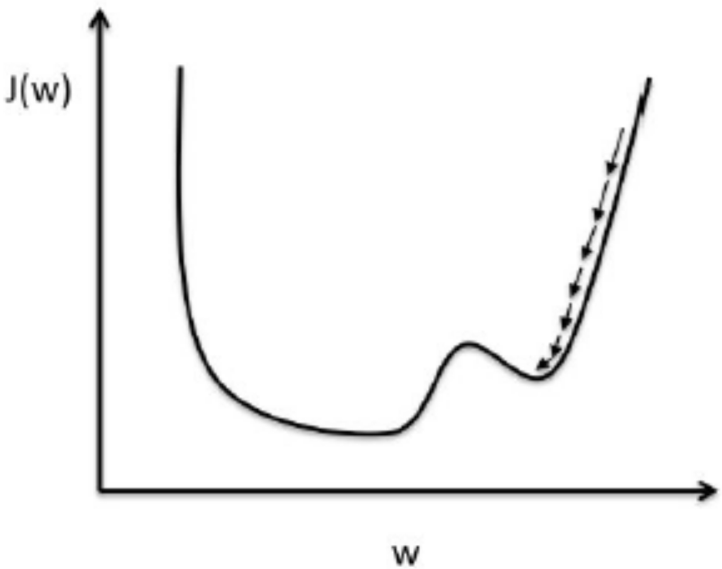
$$\theta_1 \leftarrow \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

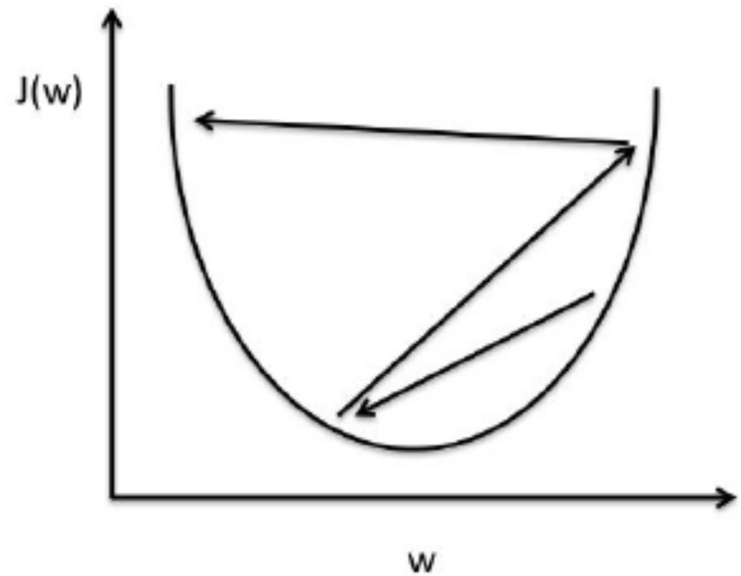
Repeat until error is small enough:

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{1}{M}\sum_{i=1}^{M}(h_\theta(x_i) - y_i)\, x_i$$

Problem: Learning Rate





Small learning rate:

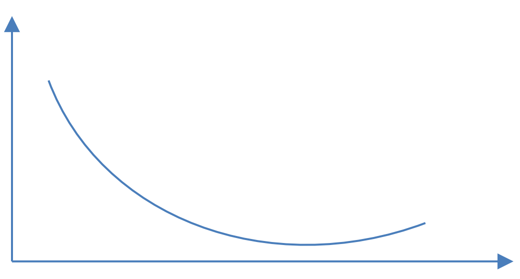
- Many iterations till convergence
- Trapped in local minimum

Large learning rate:

- Overshooting
- No convergence

Learning Rate and No. of Iterations

- Plot no. of iterations x-axis
  cost $J(\theta)$ y-axis
- If increases then decrease **learning rate** α
- Stop when $\Delta J(\theta)$ smaller than **chosen threshold**



Linear Regression (multiple variables)

- Predict single output y
  from **vector input** $\boldsymbol{x} = (x_1, \ldots, x_N)$
- Linear hypothesis h :

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

- If we add $x_0 = 1$, we can express

$$h_\theta(\boldsymbol{x}) = \boldsymbol{x}\,\theta$$

Gradient Descent (multiple variables)

Repeat until error is small enough:

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(\boldsymbol{x}_i) - y_i)\, x_{i,0}$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(\boldsymbol{x}_i) - y_i)\, x_{i,1}$$

$$\theta_2 \leftarrow \theta_2 - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(\boldsymbol{x}_i) - y_i)\, x_{i,2}$$

$$\cdots$$

$$\theta_N \leftarrow \theta_2 - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(\boldsymbol{x}_i) - y_i)\, x_{i,N}$$

$\Rightarrow$ Repeat until error is small enough:

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(\boldsymbol{x}_i) - y_i)\, x_{i,j} \quad \text{for j = 0,...,}N$$

Gradient Descent (multiple variables)

Make steps of size α (**learning rate**) down the **cost function** *J* in the **direction with the steepest descent** (as determined by **slope of the tangent** at $(\theta_0, \theta_1, \ldots \theta_N)$)

Repeat until error is small enough:

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{M} \sum_{i=1}^{M} (h_\theta(\boldsymbol{x}_i) - y_i)\, x_{i,j} \quad \text{for } j = 0,\ldots,N$$

[Or $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\alpha}{M} \mathrm{X}^{\mathrm{T}}(\mathrm{X}\boldsymbol{\theta} - \boldsymbol{y})$ in vectorized form]

Linear Regression

	1st feature	2nd feature	3rd feature	4th feature	label
	Size (sq.ft.)	**No. of Bedrooms**	**No. of Bathrooms**	**Age of Building (yrs)**	**Rent per Month ($)**
1st example	1700	4	3	10	70k
2nd example	1420	3	2	12	54k
3rd example	1290	4	1.5	8	45k
4th example	880	2	2	2	40k
5th example	510	2	2	3	26.5k

$x_{ij} = j^{\text{th}}$ feature of i^{th} example

$y_i =$ label associated with i^{th} example

Feature Scaling

- Note that the value ranges of different variables are very different, e.g., size of flat is in hundreds or thousands sq ft, age of flat is in tens.
- As α used across all *N* **features** (variables), would like input values to be roughly in same range

  $\Rightarrow$ **feature scaling** or **mean normalization**

$$x_j \leftarrow \frac{x_j - \mathrm{mean}_j}{\mathrm{max}_j - \mathrm{min}_j}$$