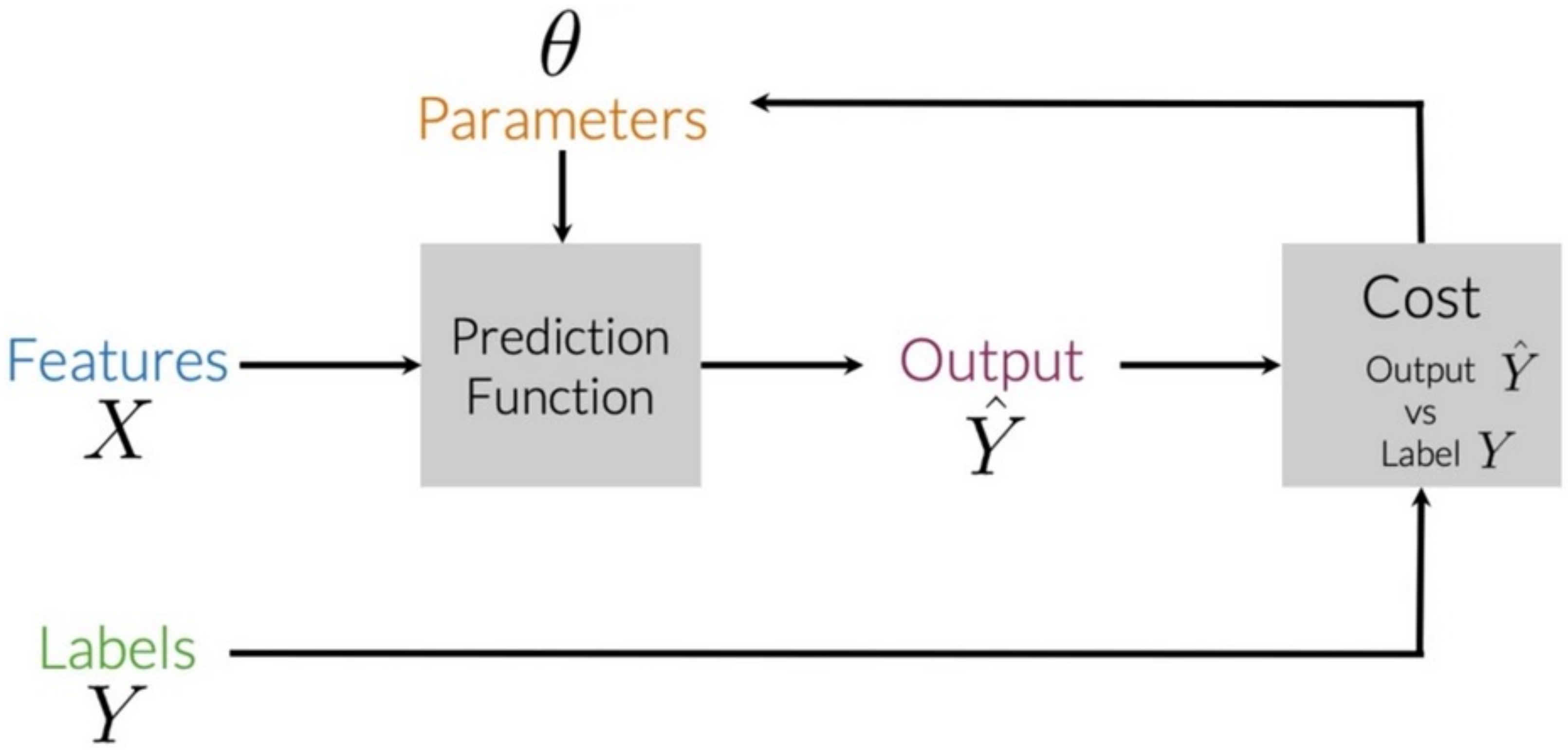
Logistic Regression

Slides/explanations from :

- Natural Language Processing with Classification and Vector Spaces
- Machine Learning with Andrew Ng
- Logistic Regression with Sebastian Raschka

Supervised Learning

- Supervised Learning Paradigm



Sentiment Analysis/Prediction

- We will be performing sentiment analysis on tweets.
- Given a tweet, our algorithm will simply have to predict if the sentiment expressed is :
  - Positive : I love NLP !
  - Negative : That new Batman movie was so bad…
- This is a **classification** task (even though the algorithm has the name *regression* in it…).
- But the model is practically the same as with linear regression, $\boldsymbol{\theta}^T \boldsymbol{x}$ is also at the heart of computing the prediction.

Representing a tweet as a vector

- How do we feed our tweet examples to the algorithm ?
- We need to find a way to convert each example into a vector !

Vocabulary and Feature Extraction

- An easy approach is to use the vocab found in the corpus.

Tweets:
[tweet_1, tweet_2, ..., tweet_m]

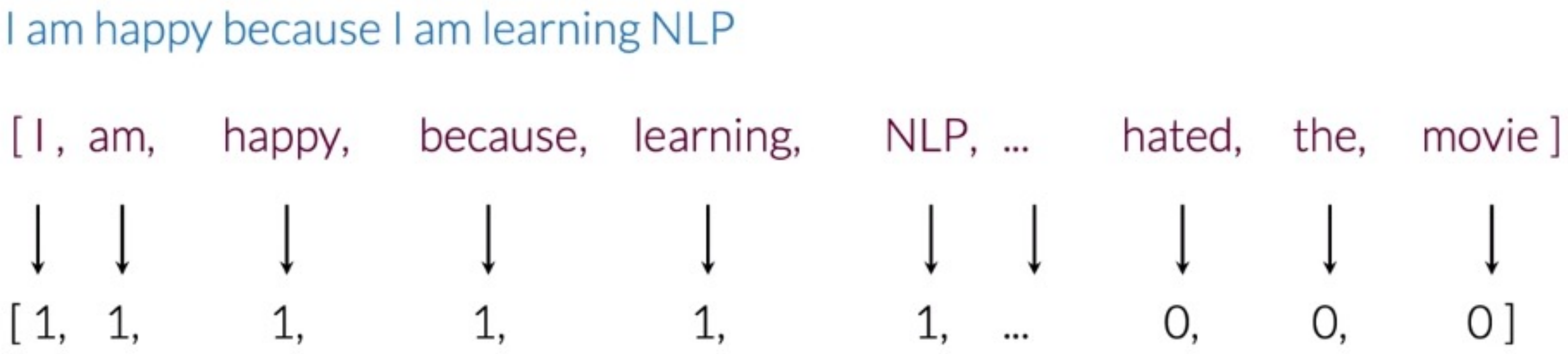
I am happy because I am learning NLP
...
...
...
I hated the movie

$V =$

[I, am, happy, because, learning, NLP, ... hated, the, movie]

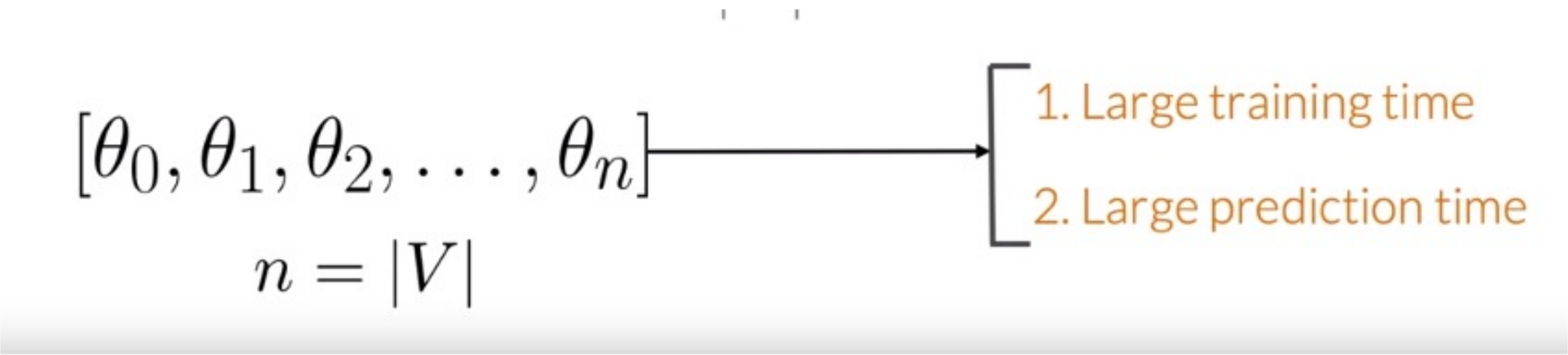
Vocabulary and Feature Extraction

- The vector for each tweet will be composed of 1s and 0s:
  - 1 for each vocab word the tweet contains
  - 0 everywhere else.



Problems with this representation

- This is called a **sparse** representation and has several problems :
  - Many of the features will be 0s
  - The vectors for each tweet will be as long as the vocab length (several thousands of dimensions…)
  - And our parameter vector $\boldsymbol{\theta}$ will also be very large



Mini Exercise

- How long would the parameter vector be if we had a corpus composed of these tweets, using the method enounced previously ? (Don’t forget about θ_0)

- I am happy to learn NLP
- I love that movie
- I love Deep Learning

Feature Extraction

- To reduce the size of our vectors and taking into account our task (sentiment prediction), let's extract some simple features that will be useful for our model !

- A basic feature we can use are the **frequencies** of words in negative/positive examples.

- The intuition is that :
  - If a tweet contains a word that is very frequent in positive tweets, then this should be a strong *positive* indicator for our model.

Positive Frequency Counts

- We create a frequency mapping for each token that occurs in positive tweets

Positive tweets

I am happy because I am learning NLP

I am happy

Vocabulary	PosFreq (1)
I	3
am	3
happy	2
because	1
learning	1
NLP	1
sad	0
not	0

Negative Frequency Counts

- Same thing for the negative tweets :

Vocabulary	NegFreq (0)
I	3
am	3
happy	0
because	0
learning	1
NLP	1
sad	2
not	1

Negative tweets
I am sad, I am not learning NLP
I am sad

Feature Extraction

- We can now use our token mappings to build vector representations for our tweets.
- For each token in the tweet, we retrieve its *positive* and *negative* frequency and sum. Each example is now represented by a vector of 3 values

$$\underbrace{X_m}_{\text{Features of tweet m}} = [\underbrace{1}_{\text{Bias}}, \underbrace{\sum_w freqs(w, 1)}_{\text{Sum Pos. Frequencies}}, \underbrace{\sum_w freqs(w, 0)}_{\text{Sum Neg. Frequencies}}]$$

Positive Feature Extraction

Feature extraction

Vocabulary	PosFreq (1)
I	3
am	3
happy	2
because	1
learning	1
NLP	1
sad	0
not	0

I am sad, I am not learning NLP

$$X_m = [1, \sum_w freqs(w, 1), \sum_w freqs(w, 0)]$$

$$\sum_w freqs(w, 1) \rightarrow \boxed{8}$$

Your turn ! Find the value for the third feature

Vocabulary	NegFreq (0)
I	3
am	3
happy	0
because	0
learning	1
NLP	1
sad	2
not	1

I am sad, I am not learning NLP

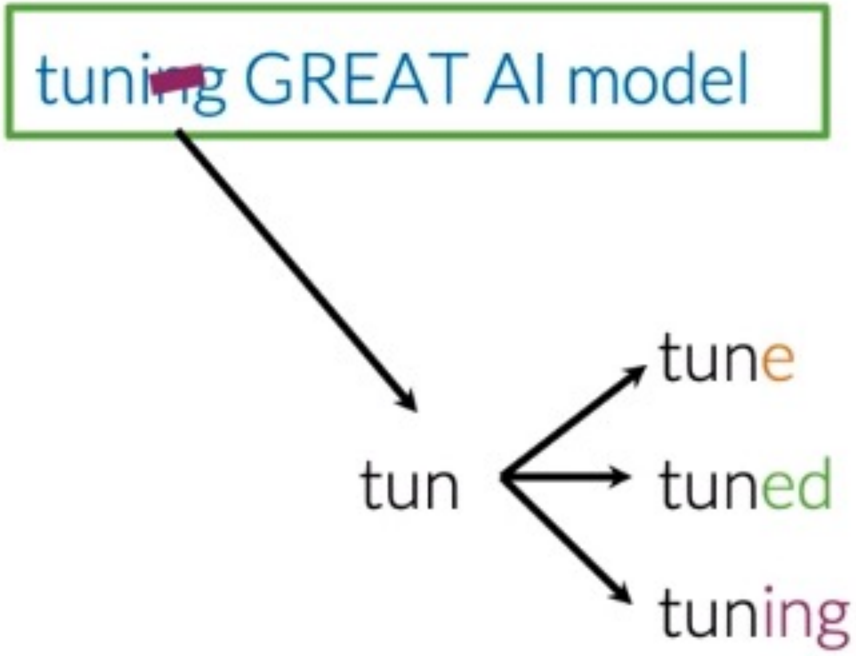
$$X_m = [1, \sum_w freqs(w,1), \sum_w freqs(w,0)]$$

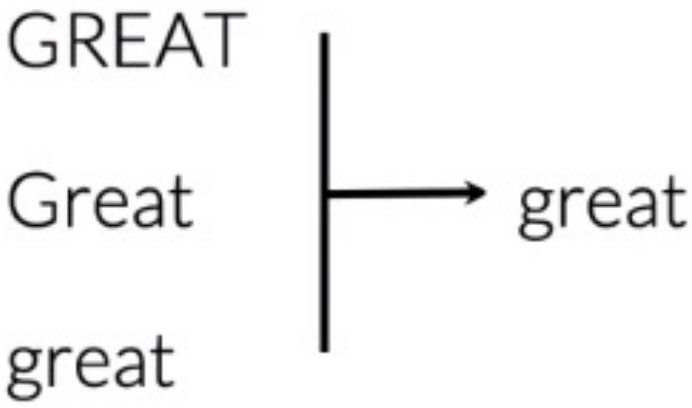
Preprocessing:

- When preprocessing text, you can eliminate:
  - Stopwords
  - Punctutation
  - Twitter handles and urls in the case of tweets…
- These steps are not always necessary, you have to decide if having these words/characters in your vocabulary is useful for your task or not…

@YMourri and @AndrewYNg are
tuning a GREAT AI model at
https://deeplearning.ai!!!

Preprocessing: Stemming and lower casing

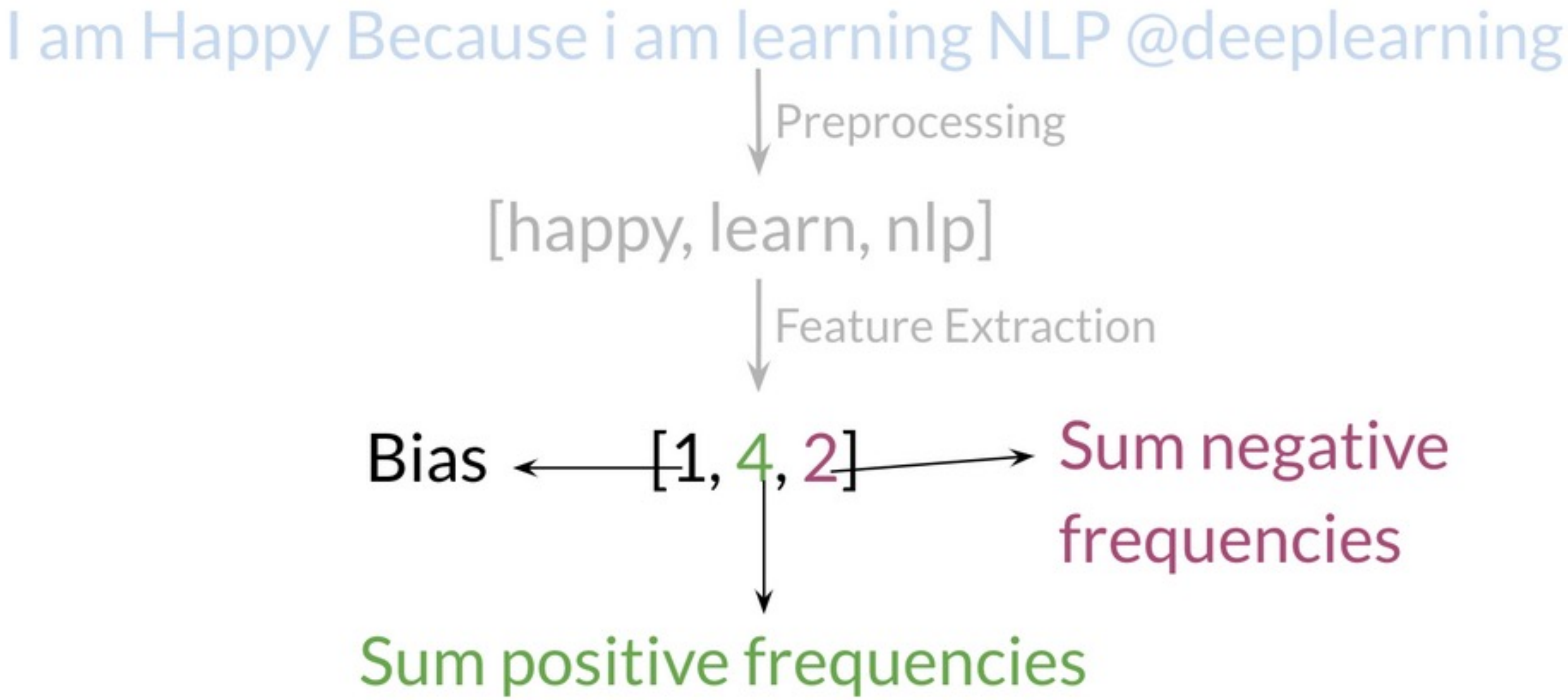




Final preprocessed tweet :

[tun, great, ai, model]

- A summary of the different steps :

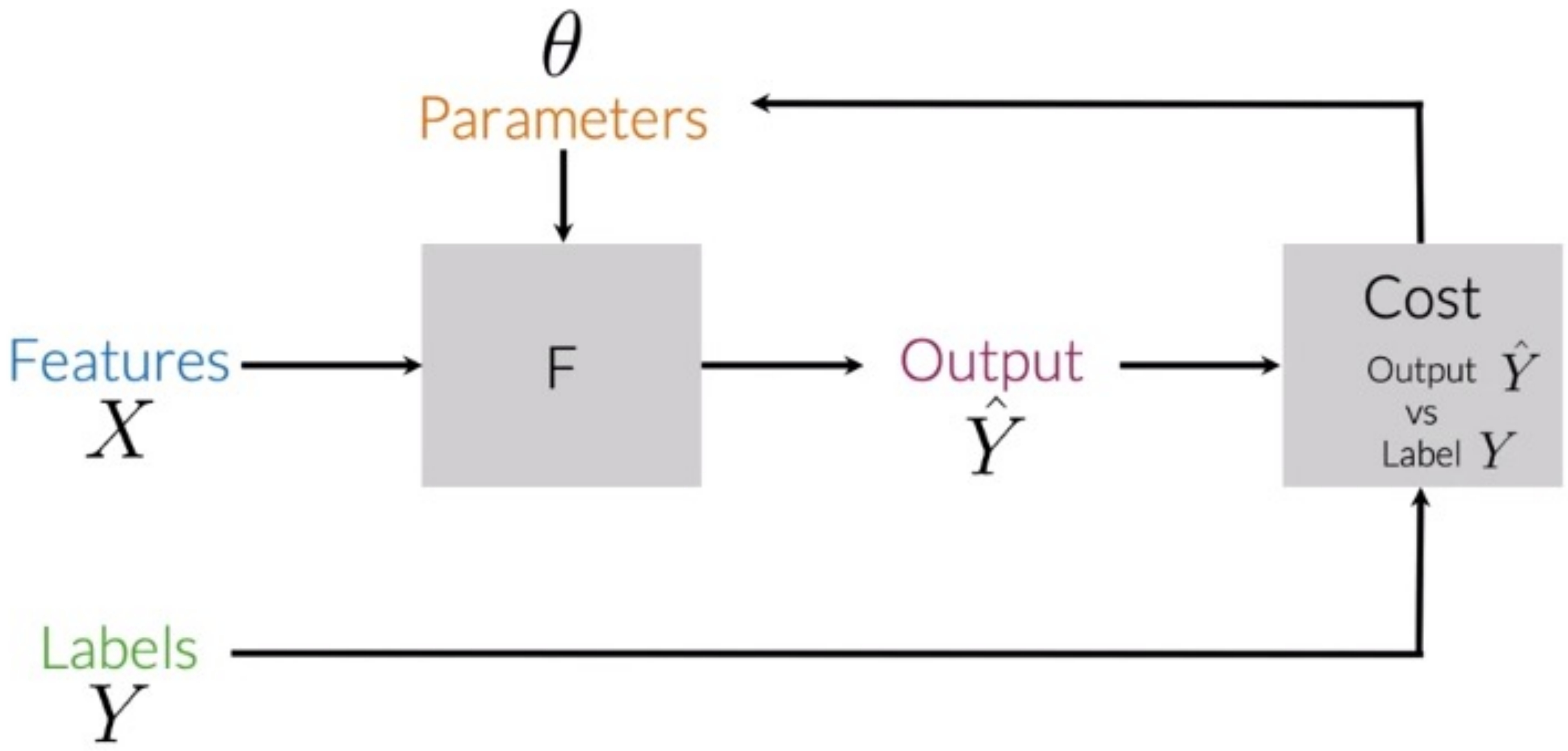


Matrix of examples X

- Instead of feeding each input vector 1 by 1, we can stack all examples into a matrix and feed it directly to the logistic regression model.
- This matrix will have the following dimensions:
- $dims = (num_{examples}, num_{features}) = (m, 3)$

$$\boldsymbol{X} = \begin{bmatrix} 1 & X_1^{(1)} & X_2^{(1)} \\ 1 & X_1^{(2)} & X_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & X_1^{(m)} & X_2^{(m)} \end{bmatrix}$$

Logistic Regression



Logistic Regression vs Linear Regression

- For linear regression, our hypothesis function was :

$$h_{linear}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ \ldots + \theta_n x_n = \boldsymbol{\theta^T} \ \boldsymbol{x}$$

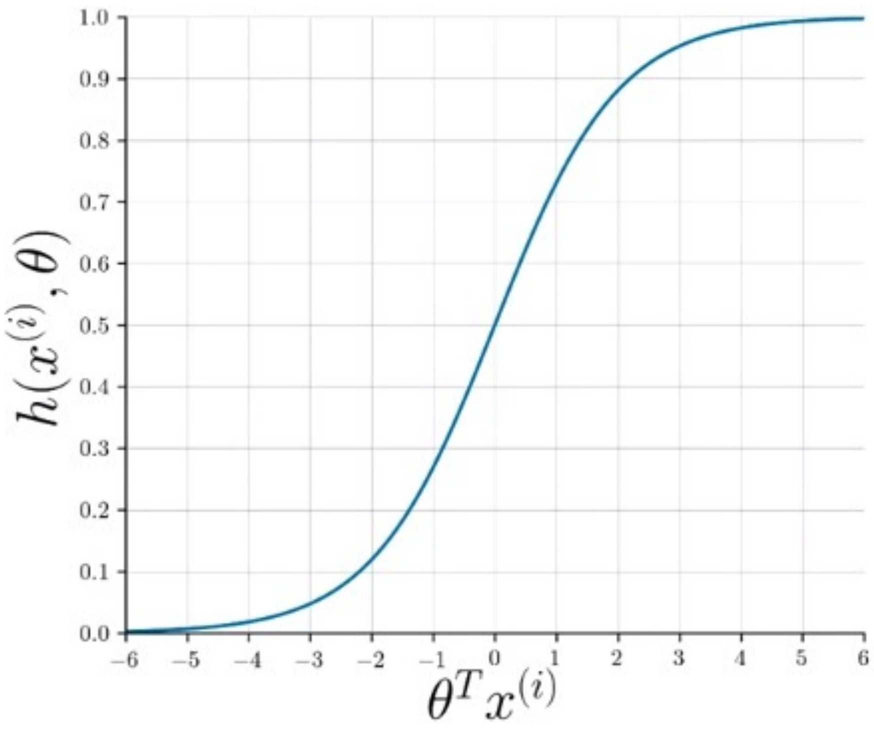
- For logistic regression, the model above is the **first** step. In the **second** step, the output goes through a special function, called the **sigmoid** / **logistic** function (this is where the model's name comes from).

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

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

The sigmoid function

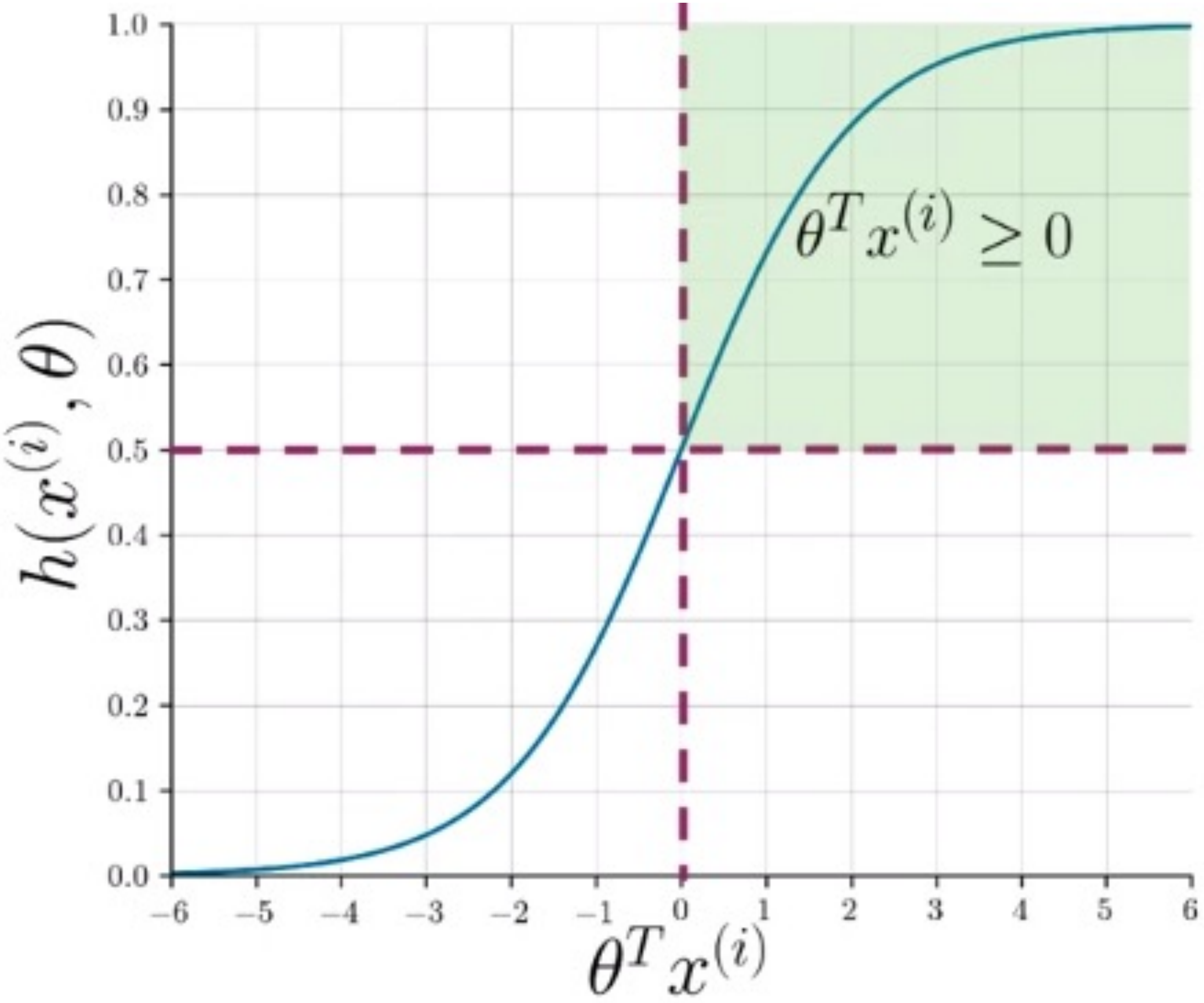
- This is the function you use when you have a binary prediction problem in machine learning => 1 out of 2 possible classes to predict.
- Here is the final formula for the model and a plot of the sigmoid function.



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

Sigmoid Intuition

- Sigmoid's output fluctuates between 0 and 1
- So the output of the model can be interpreted as a **probability.**
- And the idea is that if the output ≥ 0.5, then we classify the example as positive.
- Looking at the graph, if $\theta^T x^{(i)}$ ≥ 0, then the predicted sentiment will be Positive.



Example

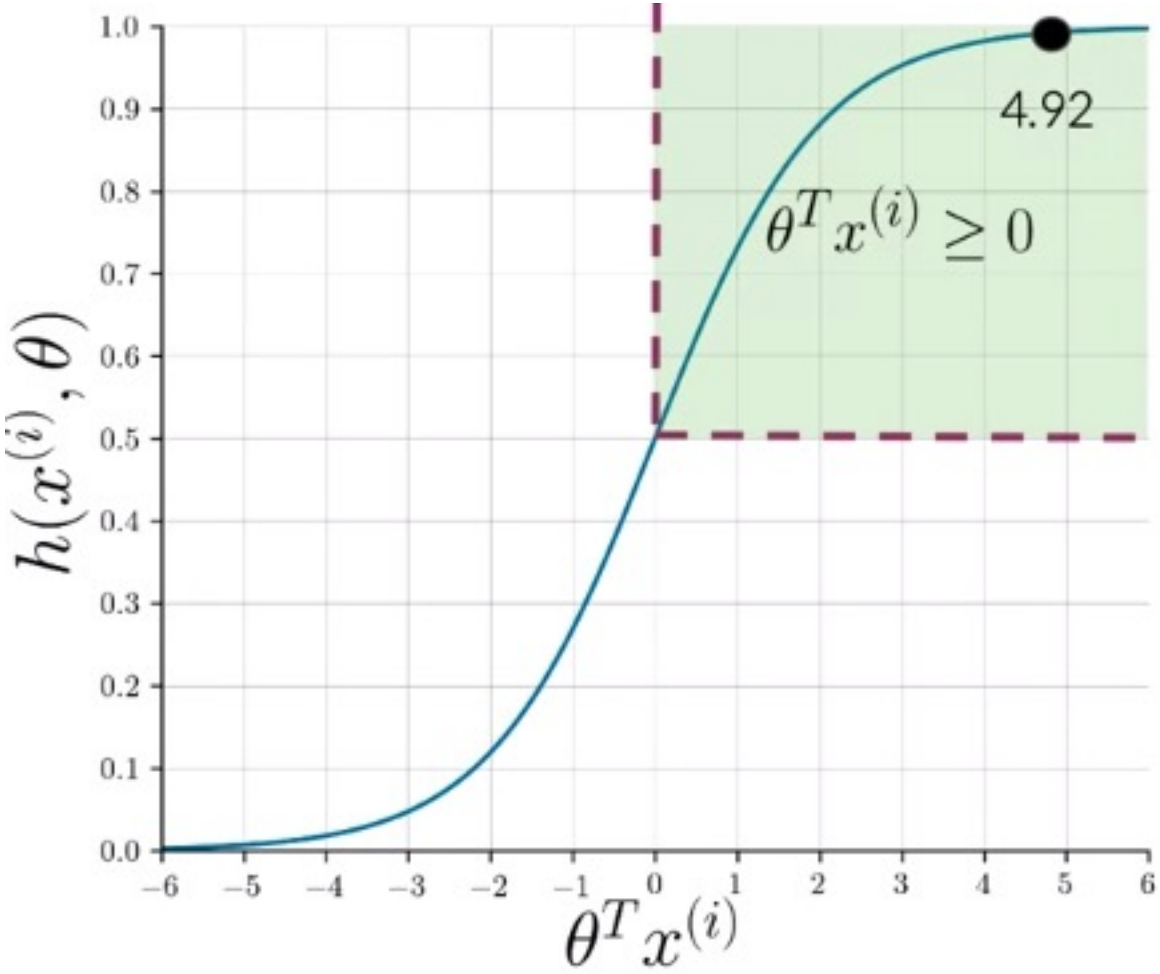
@YMourri and
@AndrewYNg are tuning a
GREAT AI model

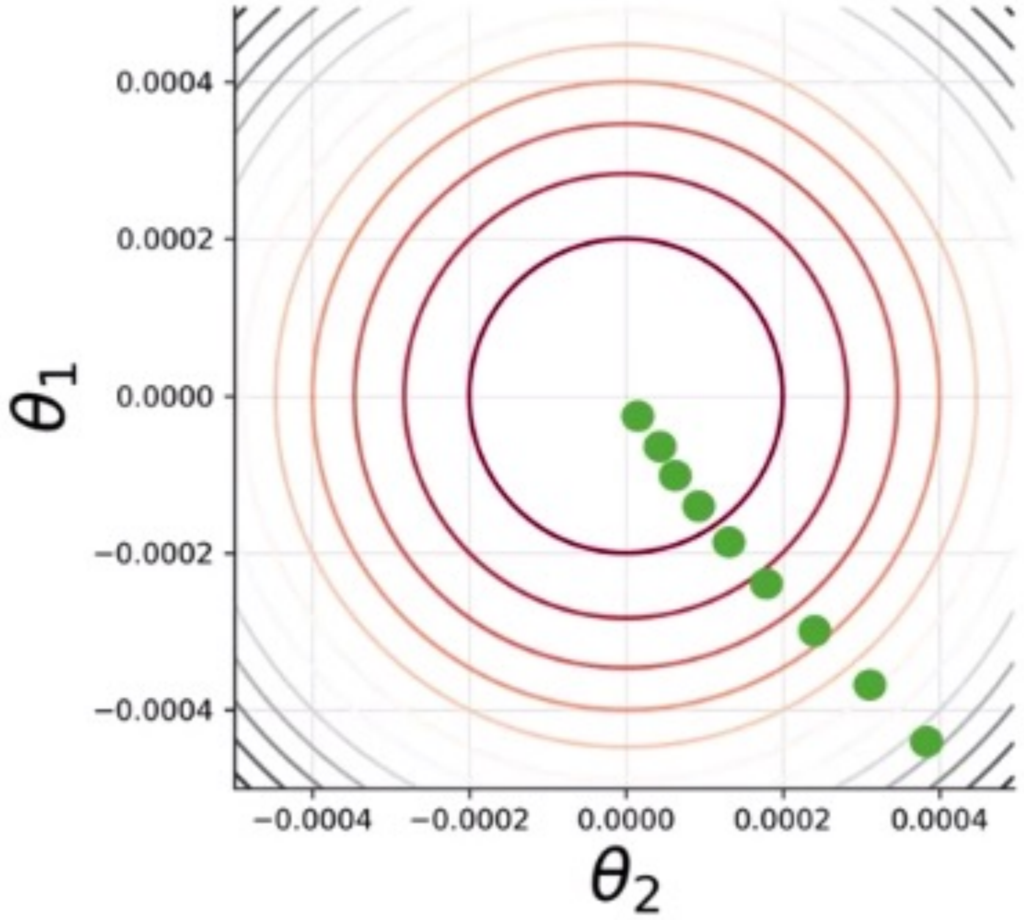
↓

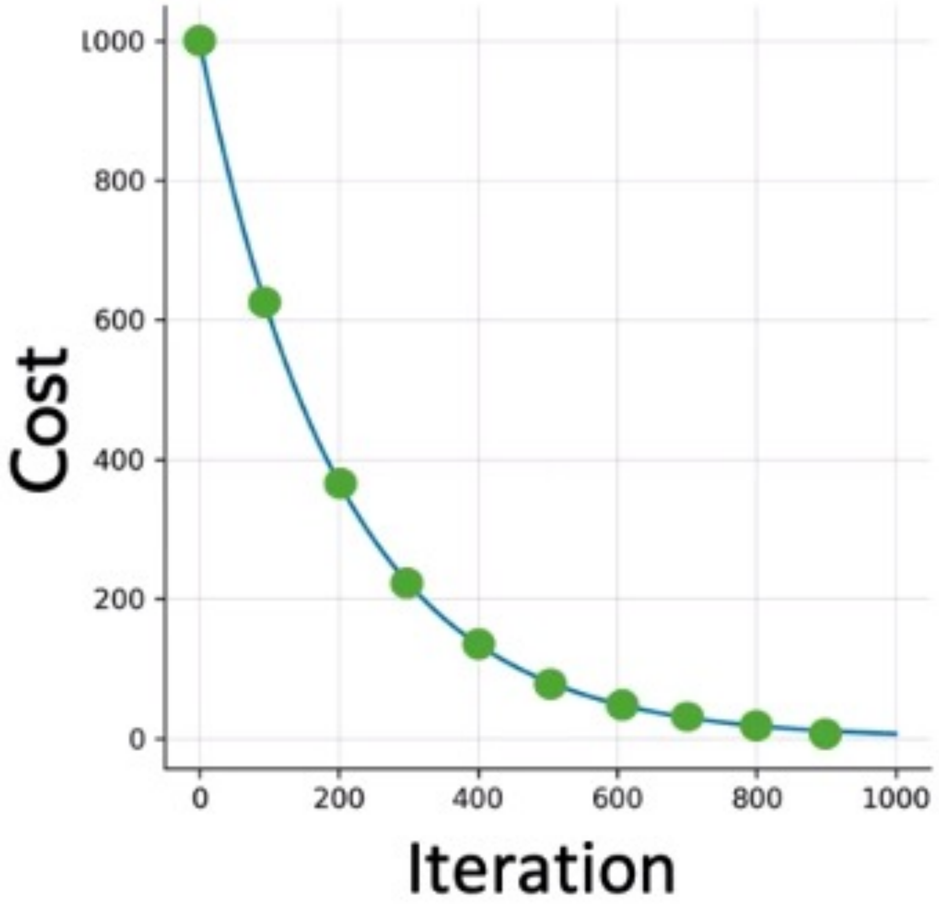
[tun, ai, great, model]

↓

$$x^{(i)} = \begin{bmatrix} 1 \\ 3476 \\ 245 \end{bmatrix} \quad \theta = \begin{bmatrix} 0.00003 \\ 0.00150 \\ -0.00120 \end{bmatrix}$$




0.0004
0.0002
0.0000
−0.0002
−0.0004
θ_1
−0.0004
−0.0002
0.0000
0.0002
0.0004
θ_2


1000
800
600
400
200
0
Cost
0
200
400
600
800
1000
Iteration

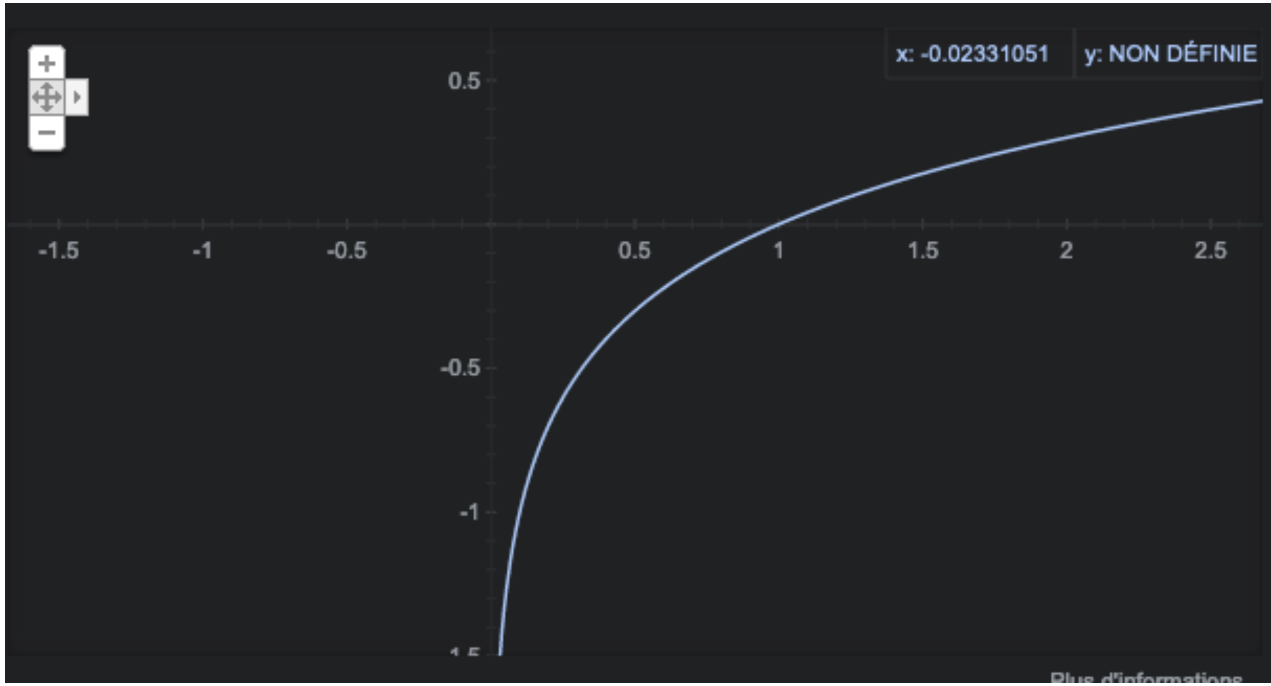
Cost Function :

- This isn't the same cost function as before (MSE) and looks like a much longer, complicated equation…. But not for long, with a little intuition.

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

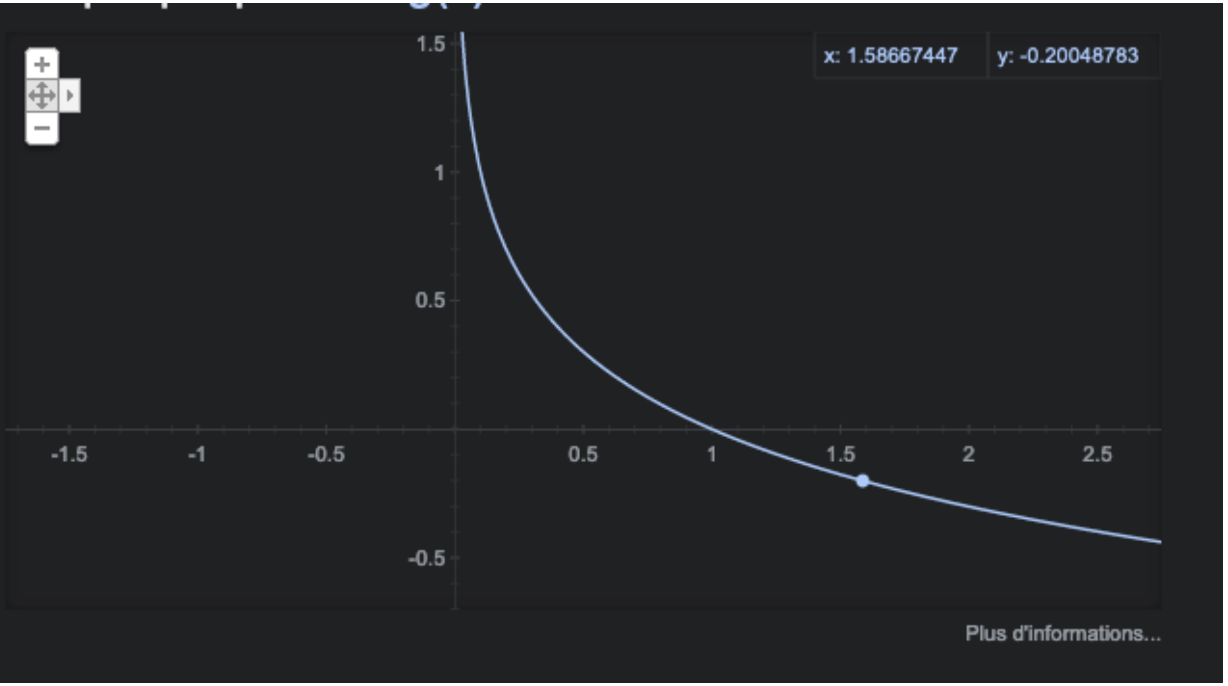
- A sum of m elements divided by m clearly looks like an average, the average error over all of our examples.
- The minus sign ensures our cost is positive, as the log of values between 0 and 1 is negative.

- Log(x)



x: -0.02331051
y: NON DÉFINIE
0.5
-0.5
-1
-1.5
-1
-0.5
0.5
1
1.5
2
2.5

- -log(x)

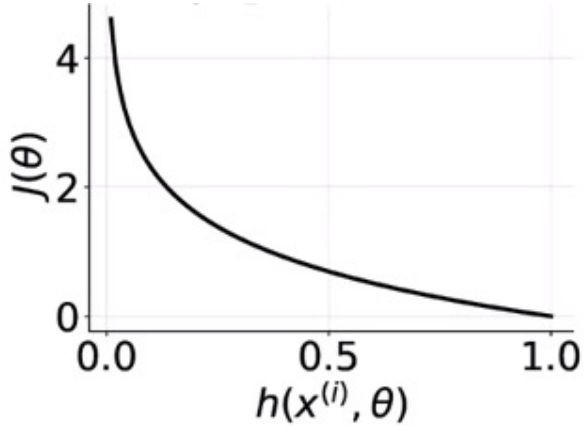


x: 1.58667447
y: -0.20048783
1.5
1
0.5
-0.5
-1.5
-1
-0.5
0.5
1
1.5
2
2.5
Plus d'informations...

2 terms: term n°1

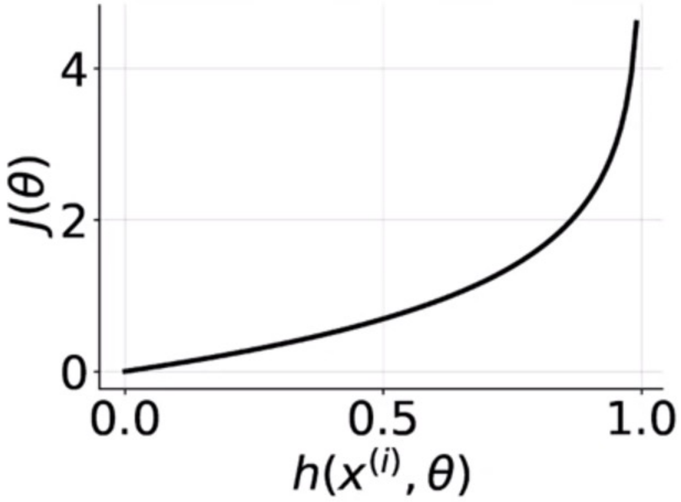
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[\boxed{y^{(i)} \log h(x^{(i)}, \theta)} + \boxed{(1 - y^{(i)}) \log(1 - h(x^{(i)}, \theta))}\right]$$

$y^{(i)}$	$h(x^{(i)}, \theta)$	Overall value
0	any	0
1	0.99	~0
1	~0	-inf



2 terms : term n°2

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



$y^{(i)}$	$h(x^{(i)},\theta)$	Overall term
1	any	0
0	0.01	~0
0	~1	-Inf

Logistic Regression: Testing

- Grab your examples from your test/validation set and trained params.

$$X_{val} \ Y_{val} \ \theta$$

- Compute probabilities $h(X_{val}, \theta)$

- Turn your probabilities into 1 & 0s, ie. Positive or Negative sentiment.

$$pred = h(X_{val}, \theta) \geq 0.5$$

Prediction vector

$$\begin{bmatrix} 0.3 \\ 0.8 \\ 0.5 \\ \vdots \\ h_m \end{bmatrix} \geq 0.5 = \begin{bmatrix} 0.3 \geq 0.5 \\ 0.8 \geq 0.5 \\ 0.5 \geq 0.5 \\ \vdots \\ pred_m \geq 0.5 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix}$$

Computing the accuracy

$$\sum_{i=1}^{m} \frac{(pred^{(i)} == y_{val}^{(i)})}{m}$$

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ pred_m \end{bmatrix} == \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ Y_{val_m} \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ pred_m == Y_{val_m} \end{bmatrix}$$

Accuracy Example

- What is the accuracy of our predictions in the example below ?

$$Y_{val} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad pred = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$