Week #2: Logistic Regression as a Neural Network

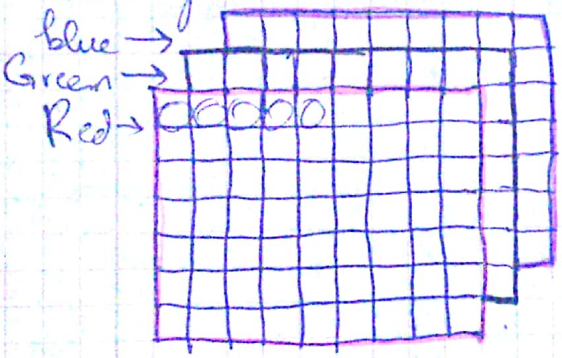
→ Logistic regression is an algorithm for binary classification.

Let's set up an example:-

Suppose we have an image and we want our output to be 1 or 0.

where 1 denotes the label we intended is true and 0 denotes otherwise.

* An image is comprised of 3 matrices in the computer, red, green and blue.



Take the matrix values and append all of them into a vector: long vector.

$$X = \begin{bmatrix} \text{red} \\ \vdots \\ \text{Green} \\ \text{blue} \end{bmatrix}$$

Suppose we have picture of 64×64

our vector will be

$$\boxed{64 \times 64 \times 3} \times \boxed{1}$$

$$12288 \times 1$$

Goal:-

we want to train a model which can take this long feature vector X and tell us whether our image represents certain intended object or not.

Notations:-

One training example :- $(x, y) \quad x \in \mathbb{R}^{n_x}, \; y \in \{0, 1\}$

Training set comprises of m training examples:-

m training examples : $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$m = m_{train}$ $\quad m_{test}$ = # test examples.

- Finally to put all of training examples into a more compact notation, we are going to define a matrix X, as defined by taking the inputs and stacking them in columns.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad \uparrow n_x \downarrow \quad \longleftrightarrow m \longrightarrow$$

$$X \in \mathbb{R}^{n_x \times m} \qquad X.shape = n_x \times m$$

- To make implementation of neural networks easier we are gonna stack y's too

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$$y \in \mathbb{R}^{1 \times m} \Rightarrow y.shape = (1, m)$$

Logistic Regression:-

- This algorithm is a supervised learning algorithm where where in the output vector either all labels are Zero or 1.

→ Given an image and then stacking all the values in a vector.

∴ Given an input vector $\vec{x}$ to an image,

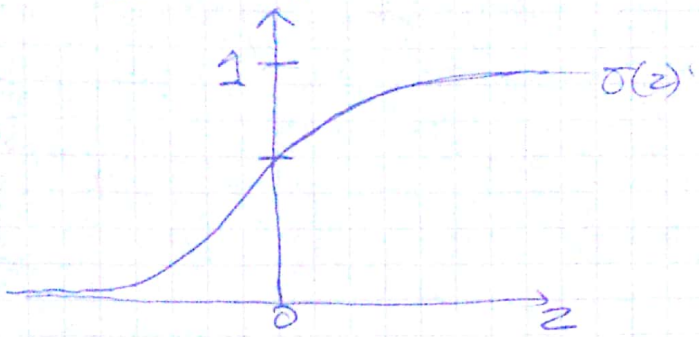
$$\boxed{\hat{y} = P(y=1 \mid x)} \qquad x \in \mathbb{R}^{n_x}$$

- Parameters of logistic regression:

$$w \in \mathbb{R}^{n_x} \;, \; b \in \mathbb{R}.$$

So, given a vector $\vec{x}$ and parameters $\vec{w}$ and b we want to know the output $\hat{y}$.

Output: $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$ $\quad y \in [0 \le y \le 1]$



$\sigma(z) = \frac{1}{1+e^{-z}}$ ⇒ If z is very large then e^{-z} will be close to zero so $\sigma(z) \approx \frac{1}{1+0} = 1$.

⇒ If z is very small or it is very large negative number then e^{-z} becomes very large and $\sigma(z)$ becomes very less

✱ When we implement the logistic regression, our job is to learn parameters w and b such that $\hat{y}$ becomes a good estimate.

Cost Function:-

$\hat{y} = \sigma(w^T x + b)$, where $\sigma(z) = \frac{1}{1+e^{-z}}$

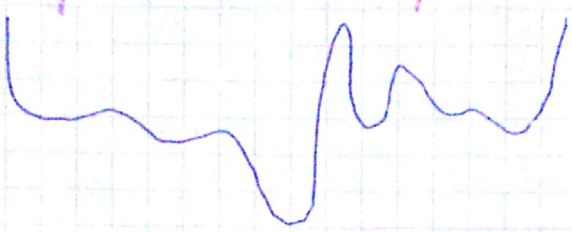
Given $\{(x^1, y^1), \ldots, (x^m, y^m)\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

✱ Now let's see how can we define a loss (error) function to measure, how well our algorithm is doing.

One thing we can do is define loss by our predicted label $\hat{y}$ and true label y as ½ half the square error.

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

- we could use this, but in logistic regression people don't normally use this, because in logistic regression our optimization problem becomes non-convex and we end up with multiple local optima.



- and in this case, gradient descent may not find the global optimum.

So in logistic regression we define another loss function that plays a similar role as squared function but gives us an optimization problem that is convex.

following is the loss function we use.

- $$L(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

% $y = 1$: $L(\hat{y}, 1) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large.

minimum loss shows that $\hat{y}$ should be as large as possible and clearly it can't be larger than 1 so in other words we want $\hat{y}$ as close as to 1

- If $y = 0$ $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ to be large and thus want $\hat{y}$ as small as possible.

→ The loss function was defined w.r.t one training example, it measures, how well you are doing on a single training example.

Now lets define Cost function which measures how well you are doing on the entire training examples.

Cost function:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

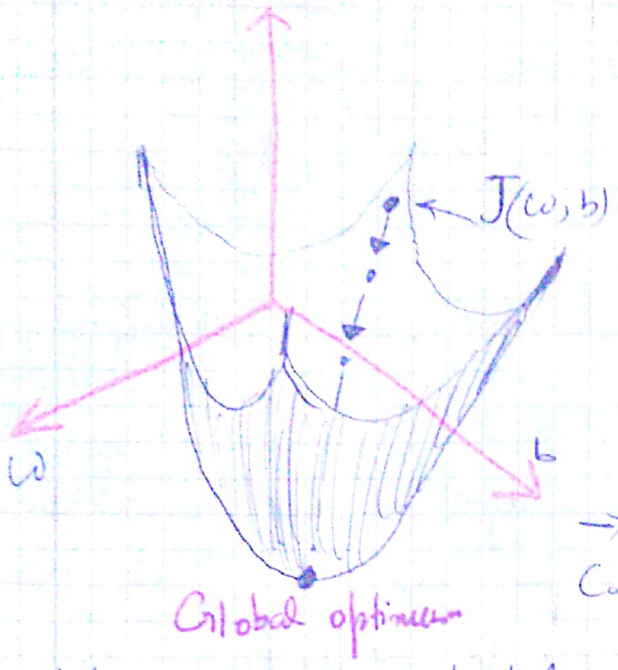
Loss function is applied to single training example and the cost function is the cost of your parameters, so we want to find the parameters which minimizes the cost function J.

Gradient Descent:- Lets talk about, how can we use gradient descent on our algorithm to learn the parameters w and b

Recap: $\hat{y} = \sigma(w^T X + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m} y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\log\hat{y}^{(i)})$$

We want to find w, b that minimizes $J(w,b)$.

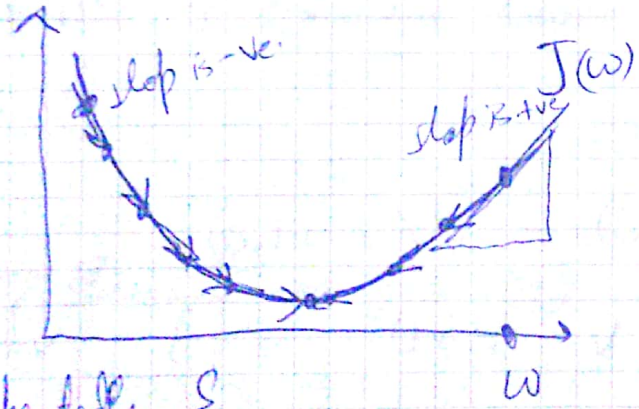


⇒ In practice w can be of higher dimension but for the sake of this example let's take them as one scalar

→ Cost function J is a Convex function

What we do, is initialize w and b to some random value like the blue dot on the graph after some iterations we converge to our global optimum

Illustration:



Let's say, we have a function that we want to minimize

Repeatedly {

$$w := w - \alpha \frac{dJ(w)}{dw}$$

Update on parameters.

until the algorithm converges

}.

$\alpha \Rightarrow$ learning rate

$J(w, b)$ $\quad w := w - \alpha \frac{dJ(w,b)}{dw}$ ← partial derivative.

$b := b - \alpha \frac{dJ(w,b)}{db}$

Computational graph:-

→ Computations of a neural networks are organized in terms of a forward pass or a forward propagation step, in which we compute the output of the Neural Network followed by the backward pass or backward propagation step, which we use to compute gradients.

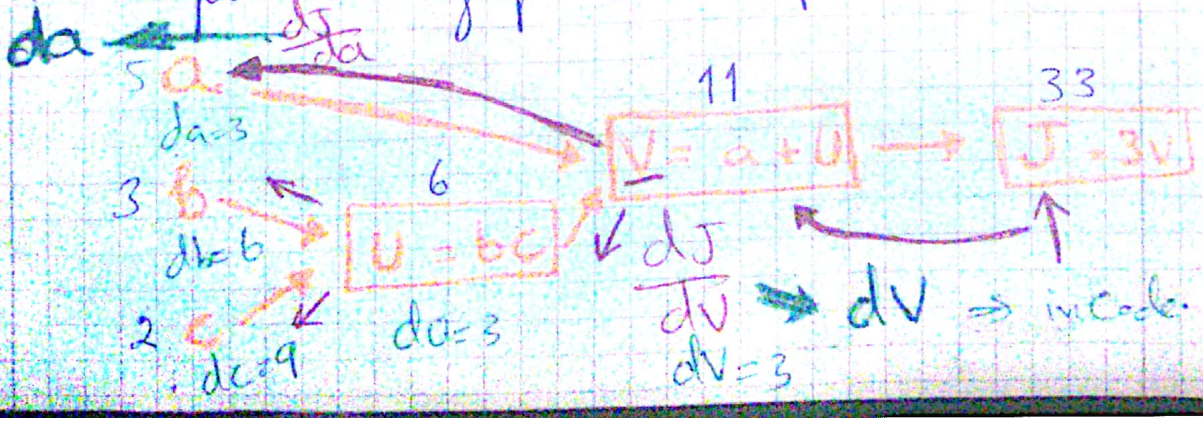
Computational graph explains this, why it is organized this way.

→ Let's use a simple example of logistic regression or a full blown neural network to illustrate Computational graph.

eg $\quad J(a, b, c) = 3(a + bc)$

① $U = bc$, ② $V = a + U$; ③ $J = 3V$

We can take these 3 steps and draw them in a computational graph as follows:-



So, the computation graph comes in handy when there's some distinguished or some special output variables such as J in this case that needs to be optimized, and in case of logistic regression, J is of course the cost function that we are trying to minimize

Derivatives with computation graphs:-

→ Keeping in consideration the last computation graph and let's say we want to compute derivative of J with respect to V $\frac{dJ}{dV}$, so that means; if we were to change the value of V, how would it affect J.

$$J = 3V \quad ; \quad V = 11 \rightarrow 11.001$$
$$J = 33 \rightarrow 33.003$$

$$\frac{dJ}{dV} = 3$$

Now what is $\frac{dJ}{da}$

$$a = 5 \rightarrow 5.001$$
$$V = 11 \rightarrow 11.001$$
$$J = 33 \rightarrow 33.003$$

→ by changing a $\frac{dV}{da}$, you end up changing V and change in the value of V will cause the value of J to also change. In calculus this is called the Chain Rule

⟶ if a affects V which in turn affects J then the amount that J changes is the product of how much V changes when you nudge a and how much J changes when you nudge V.

$$\frac{dJ}{da} = \frac{dJ}{dV} \cdot \frac{dV}{da}$$

What is $\frac{dJ}{du} = 3 = \frac{dJ}{dv}\cdot\frac{dv}{du}$

$u = 6 \rightarrow 6.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$\frac{dJ}{db} = \frac{dJ}{du}\cdot\frac{du}{db} = 6$

$b = 3 \rightarrow 3.001$

$u = bc = 6 \rightarrow 6.002 \quad c = 2$

$J = 33.006$

$\frac{dJ}{dc} = \frac{dJ}{du}\cdot\frac{du}{dc} = 9$

$3 \cdot 3$

Logistic Regression Gradient Descent:-

Logistic Regression recap:

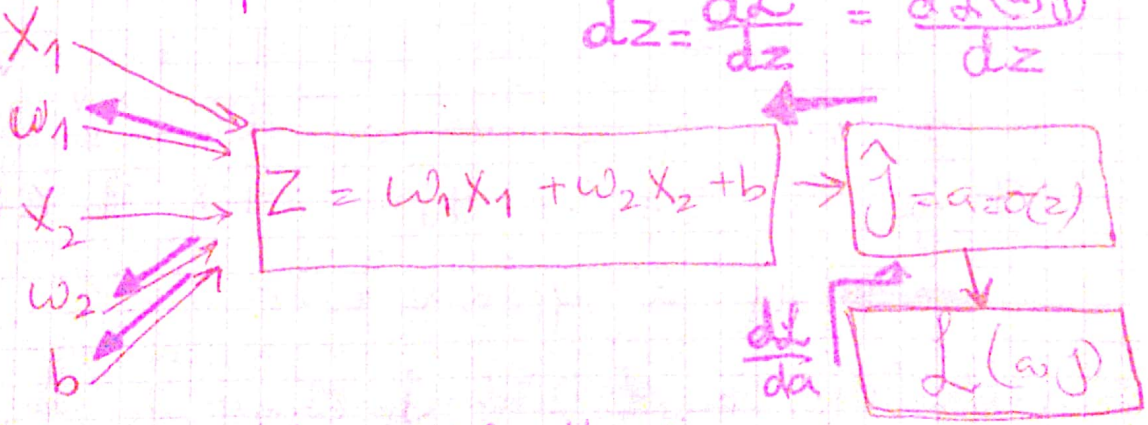
$z = w^T x + b$

$\hat{y} = a = \sigma(z)$ output

Loss function: $\mathcal{L}(a, y) = -(y \log(a) + (1-y)\log(1-a))$

For the sake of this example let's say we have two features x_1 and x_2.

So the output will be:



→ So, our job is to modify the parameters w_1, w_2 and b in order to reduce the loss function $\mathcal{L}(a, y)$

→ First thing you do is compute the derivative of loss "$\frac{d\mathcal{L}(a,y)}{da}$" or da.

So let's do that :-

$$\frac{d\mathcal{L}(a,y)}{da} = \frac{d}{da}\left[-(y\log(a) + (1-y)\log(1-a))\right]$$

$$da = -\frac{y}{a} + \frac{1-y}{1-a}$$

2nd step

$$\frac{dL}{dz} = \frac{dL}{da}\cdot\frac{da}{dz} \Rightarrow a(1-a)$$

$$a = \sigma(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{da}{dz} = +\frac{e^{z}}{(1+e^{-z})^2}$$

$$\frac{da}{dz} = \frac{1}{1+e^{-z}}\cdot\left[1 - \frac{1}{1+e^{-z}}\right] \Rightarrow a(1-a)$$

$$\frac{dL}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{dL}{dz} = -y(1-a) - a(1-y)$$

$$= -y + ya + a - ya = a - y$$

$$\frac{dL}{dz} = a - y$$

Final step: go back to check how much do you need to change w_1, w_2, b.

$$\frac{\partial \mathcal{L}_1}{\partial w_1} = dw_1 = x_1\cdot dz \quad dw_2 = x_2\cdot dz \quad db = dz$$

Now we perform update

$$w_1 := w_1 - \alpha\, dw_1$$
$$w_2 := w_2 - \alpha\, dw_2$$
$$b := b - \alpha\, db$$

Logistic Regression with m examples

Cost function:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y)$$

$$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

dw_1, dw_2, b

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w_1}\left(\mathcal{L}(a^{(i)}, y^{(i)})\right)$$

Algorithm

Let's make all of this compact into algorithm

→ Start with $dw_1 = 0$; $dw_2 = 0$; $db = 0$

$dw_1 = \frac{\partial J}{\partial w_1}$

for i = 1 to m:

$z^{(i)} = w^T x^{(i)} + b$

$a^{(i)} = \sigma(z^{(i)})$

$J \mathrel{+}= -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$

$dz^{(i)} = a^{(i)} - y^{(i)}$

another loop $dz^{(i)}$

$dw_1 \mathrel{+}= x_1^{(i)} dz^{(i)}$

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$

$db \mathrel{+}= dz^{(i)}$

assuming there are two features that's why 2 w's

$J /= m$ $dw_1 /= m$ $dw_2 /= m$ $db /= m$

Now Comes the update of gradient descent.

$$w_1 := w_1 - \alpha\, dw_1$$

$$w_2 := w_2 - \alpha\, dw_2$$

$$b := b - \alpha\, db$$

→ Now problem with this algorith is that we need two for loops to accomplish the task, one is the main loop over all the training examples other is for the updation of all w's i.e. parameters of logistic regression.

Vectorization is used to avoid all this expensive Computation, let's talk about that.

→ What is Vectorization?

Vectorization is an art of getting rid of for loops from your code.

→ In logistic regression we compute $\boxed{z = w^T x + b}$

$$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \; ; \; x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

$w \in \mathbb{R}^{n_x}$

$x = \mathbb{R}^{n_x}$ vectors.

Non-vectorized:

```
z = 0
for i in range(nx):
    z += w[i] * x[i]
z += b
```

Vectorized:

```
z = np.dot(w, x) + b
```

$w^T x$

→ More Vectorization examples:

* Whenever possible avoid explicit for loops

$$\vec{u} = A\vec{v}$$

$$u_i = \sum_j A_{ij} v_j$$

```
u = np.zeros(n,1)
for i = 1 .... :
    for j = 1 .... :
        u[i] += A[i][j] * v[j]
```

Vectorized

```
u = np.dot(A, v)
```

another example:

Say you need to apply the exponential operation on every element of a matrix/vector.

$$\vec{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \Rightarrow u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

Vectorized implementation

```
import numpy as np
u = np.exp(v)
```

Let's see how can we get rid of any of the one for loops from our example or algorithm we developed.

→ Second loop can be avoided:

Instead of defining $dw_1 \dots dw_n$ as individual variables, introduce them as a vector dw and instead individual updates, update them by using vector operations.

$$dw \mathrel{+}= x^{(i)} dz^{(i)}$$

$$dw \mathrel{/}= m$$

Vectorizing logistic Regression:-

→ Let's first examine the forward propagation step of logistic Regression.

Then from m training examples you need to do the following for 1st example:-

$$z^{(1)} = w^T x^{(1)} + b$$
$$a^{(1)} = \sigma(z^{(1)})$$

for second then ↓

$$z^{(2)} = w^T x^{(2)} + b$$
$$a^{(2)} = \sigma(z^{(2)})$$
$$\vdots$$
$$z^{(m)} = w^T x^{(m)} + b$$
$$a^{(m)} = \sigma(z^{(m)})$$

X is a feature matrix for all the training examples we have.

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \in (n_x \times m) \quad \mathbb{R}^{n_x \times m}$$

$$[z^{(1)} \; z^{(2)} \; \cdots \; z^{(m)}] = w^T X + [b \; b \; \cdots \; b]_{1 \times m}$$
$$= [w^T x^{(1)} + b \quad w^T x^{(2)} + b \quad \cdots \quad w^T x^{(m)} + b]$$

how python sees it:

$$Z = np.dot(w.T, X) + b$$

Now b is a real number, but remember in this condition python expands it and it becomes like [b b ... b]

This is called Broadcasting.

Now next thing is to compute $a^{(1)}, a^{(2)} \cdots a^{(m)}$

$$A = [a^{(1)} \; a^{(2)} \; \cdots \; a^{(m)}] = \sigma(Z)$$

Vectorizing Logistic Regression (Gradient Output):-

We computed $dz^{(1)} = a^{(1)} - y^{(1)}$, $dz^{(2)} = a^{(2)} - y^{(2)}$ ....

To vectorize, we define a new variable

$$dZ = \underset{1 \times m}{\left[dz^{(1)} \; dz^{(2)} \; \cdots \; dz^{(m)} \right]}$$

$$A = \left[a^{(1)} \; \cdots \; a^{(m)} \right]$$

$$Y = \left[y^{(1)} \; \cdots \; y^{(m)} \right]$$

$$dZ = A - Y$$

In the previous implementation:

$dw = 0$

$dw_1 \mathrel{+}= x^{(1)} dz^{(1)}$

$dw_2 \mathrel{+}= x^{(2)} dz^{(2)}$

$\vdots$

$dw_n \mathrel{/}= m$

$db = 0$

$db \mathrel{+}= dz^{(1)}$

$\vdots$

$db \mathrel{+}= dz^{(m)}$

$db \mathrel{/}= m$

Vectorized Implementation:-

$$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$$

$$= \frac{1}{m} \text{np.sum}(dz)$$

$$dw = \frac{1}{m} X dz^{T}$$

$$= \frac{1}{m} \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$dw = \frac{1}{m} \left[x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)} \right]$$

Summing all this up and Let's concoct up the vectorized version of algorithm.

```
for i in range(1000):
    Z = w^T X + b
      = np.dot(w.T, X) + b
    A = σ(Z)
    dZ = A - Y
    dw = 1/m X dZ^T
    db = 1/m np.sum(dZ)
    w = w - α dw
    b = b - α db
```

Single iteration of Gradient descent

Broadcasting in Python:-

Let's motivate broadcasting by an example:-

→ Calories from Carbs, Protiens, Fats in 100g of different foods

	Apple	Beef	Eggs	Potatoes
Carbs	56.0	0.0	4.4	68.0
Protien	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$A =$

, goal is to calculate % of Calories from Carb, protien and Fat. Can you do this with explicit for loop.

```
>> cal = A.sum(axis=0)
   Print(cal)    % [59.  239.  155.4  79.]
   Per = 100*A/cal.reshape(1,4)
   Print(Per)
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix}$$

(m, n) $\quad$ $(1, n) \rightsquigarrow (m, n)$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix}$$

if (m, n) $\overset{+}{-}$ $(1, n) \rightsquigarrow (m, n)$ then operation element wise.

If Conversely.

(m, n) $\overset{+}{-}$ $(m, 1) \rightsquigarrow (m, n)$ then elementwise operation

Note on Python / Numpy codes

$a = np.random.randn(5, 1)$ $\Rightarrow$ Preffered

$a = np.random.randn(5)$
$a.shape = (5,)$
rank 1 array

} don't use.

Logistic Regression Cost function:-

$\hat{y} = \sigma(w^T x + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

Interpret $\hat{y} = P(y=1 \mid x)$

if $y=1$: $P(y|x) = \hat{y}$

if $y=0$: $P(y|x) = 1 - \hat{y}$

Let's summerize the above 2 equations into a single one.

$$P(y|x) = \hat{y}^{y} (1-\hat{y})^{(1-y)}$$

we can write the probability as follows, do u why?
Let's check out.

1st case. if $y=1$ $P(y|x) = (\hat{y})^1 \cdot (1-\hat{y})^{(1-1)} = \hat{y}$

2nd case. if $y=0$ $P(y|x) = (\hat{y})^0 \cdot (1-\hat{y})^{(1-0)} = 1-\hat{y}$

because log function is strictly monotonically increasing function we are maximizing $P(y|x)$

$$\log P(y|x) = \log[\hat{y}^y (1-\hat{y})^{(1-y)}]$$

$$= \log(\hat{y}^y) + \log(1-\hat{y})^{(1-y)}$$

$$\boxed{\log[P(y|x) = y \log(\hat{y}) + (1-y)\log(1-\hat{y})}$$

and it is equal to the negative of the loss function.

This is equal to the negative of the loss function. because in the logistic regression we are trying to minimize the loss by increasing the probability of doing so.

So $-L(\hat{y}, y)\downarrow = \log[P(y|x)]\uparrow$

Cost on m examples:-

$$\log P(\text{labels in training set}) = \log \prod_{i=1}^{m} P(y^{(i)}|x^{(i)})$$

$$= \sum_{i=1}^{m} \log(P(y^{(i)}|x^{(i)}))$$

Maximum likelihood estimate

$$= -L(\hat{y}^{(i)}, y^{(i)})$$

log P(labels in training set)

$$= -\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Maximum likelihood estimation.

Cost: $J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

minimize