→ AI is new electricity (

→ as important as electricity revolution was

Courses

→ Neural Networks & Deep Learning

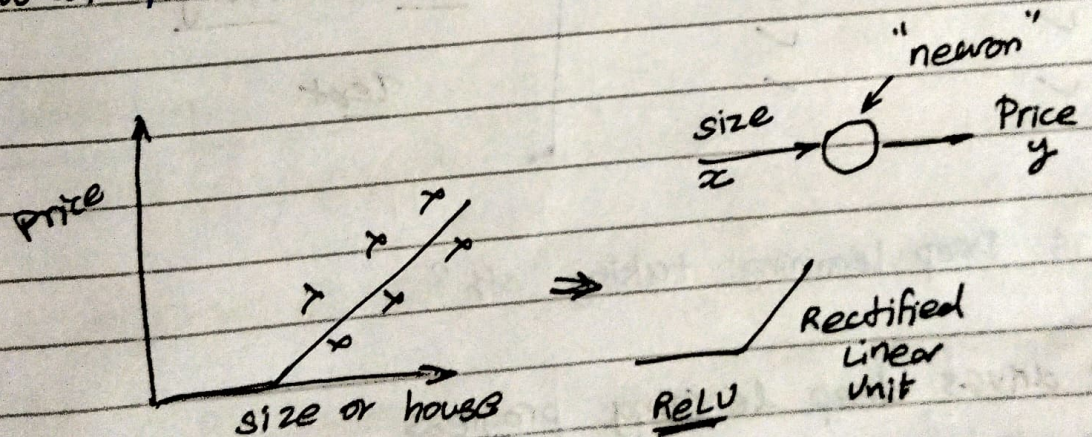
→ Improving deep learning Neural networks (Praetical)

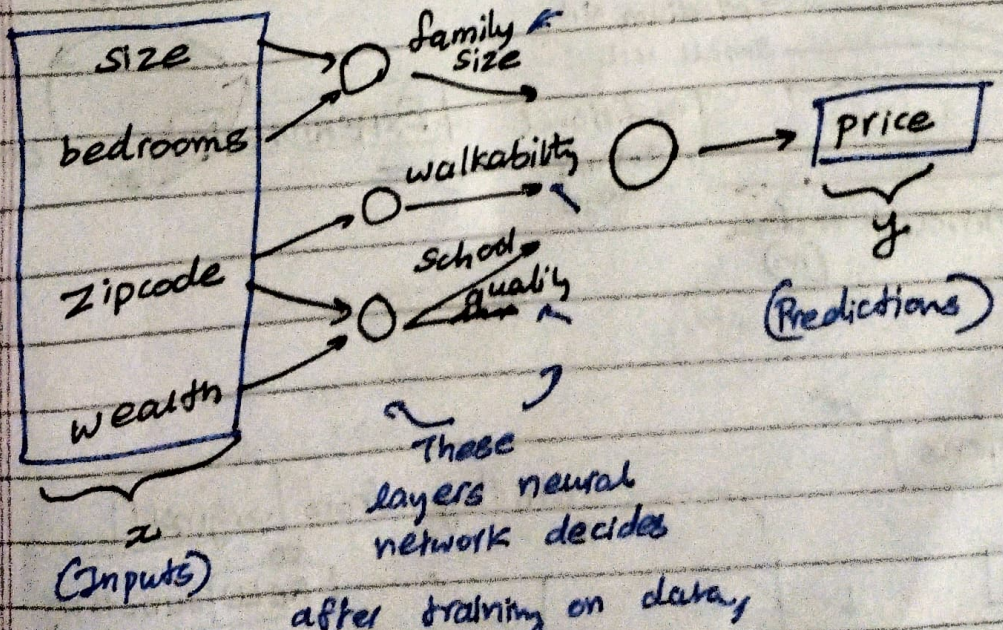
→ Structuring your Machine Learning project

→ Convolutional Neural Networks.

→ Natural Language Processing: Building sequence models

★ What is a Neural Network?





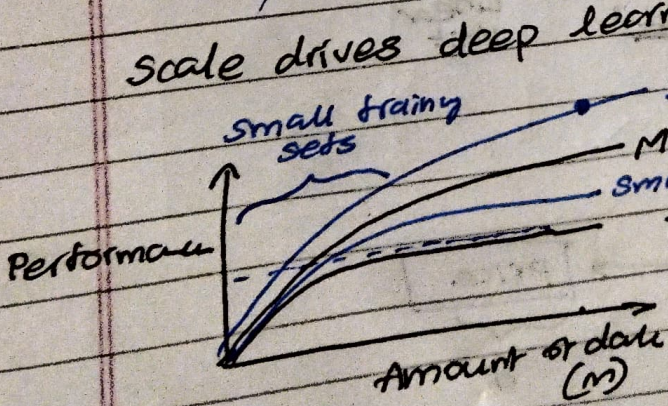
Supervised learning with neural networks.

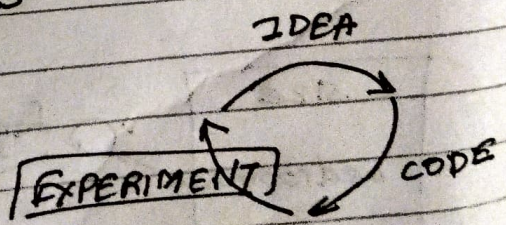
	INPUT (x)	OUTPUT (y)	
Standard NN	Home features	price	Real Estate
	Adv user info	clik on ad?	online Advertising
CNN	Image	object (1 ... 1000)	Photo Tagging
RNN	Audio	Text transcript	Speech Recognition
	English	chinese	Machine Translation
Custom/ Hybrid	Image, Radar Info	Position of other cars	Autonomous driving

Structured Data			Unstructured Data
Size	# bedrooms	Price	Audio, Image, Text
fixed	✓	✓	
✓	✓	✓	
✓	✓		
✓			

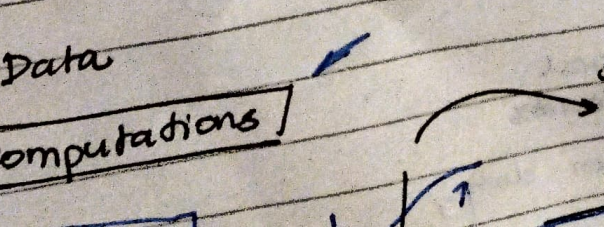
Why is Deep learning taking off?

Scale drives deep learning progress



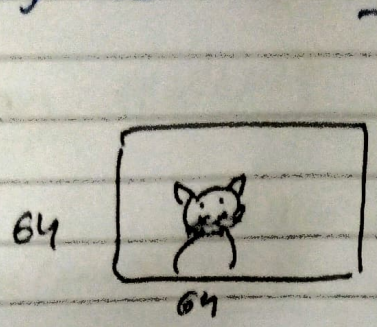


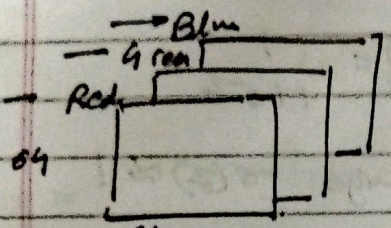
- Data
- computations



Logistic Regression as a Neural Network

Binary classification





$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ 255 \\ 134 \\ \vdots \\ 255 \\ 134 \end{bmatrix} \begin{matrix} \} \text{Red} \\ \\ \\ \} \text{Green} \\ \\ \\ \} \text{Blue} \end{matrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$x \longrightarrow y$

Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}, y \in \{0, 1\}$

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

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

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \text{ (rows)}, \quad m \text{ (columns)}$$

much easier

$X \in \mathbb{R}^{n_x \times m}$ $\quad X.shape = (n_x, m)$

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

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

Logistic Regression

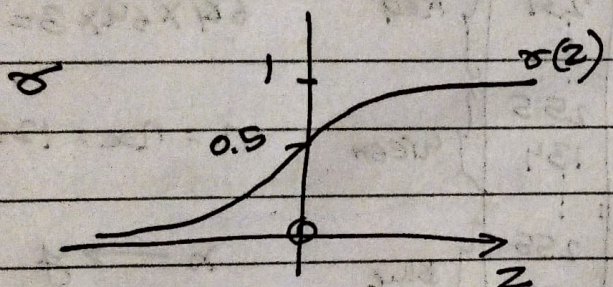
Given x, want $\hat{y} = P(y=1|x)$

$0 \le \hat{y} \le 1$

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

Parameters: $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

output: $\hat{y} = \sigma(w^T x + b)$



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

If z large $\sigma(z) \approx 1$

If z large negative number

$\sigma(z) = \frac{1}{1+\infty} \approx 0$

Different convention

$x_0 = 1$, $x \in \mathbb{R}^{n_x+1}$

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

← Not using these.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} b \leftarrow \\ \} w \leftarrow \end{matrix}$$

Logistic Regression 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 $\hat{y}^{(i)} \approx y^{(i)}$

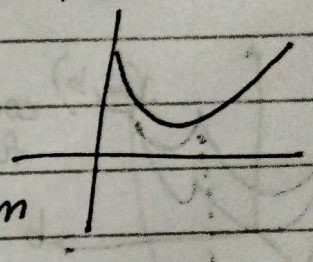
(you should have got; that you got)

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

⇒ Loss (error) function: (single training example)

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

(output; end up with optimisation problem)



So this is used ↳

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

$\hat{y}$ → close

If $y=1$ $L(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ lar

If $y=0$ $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large ... wa

$L(\hat{y}, y)$ → should be as small as possible

⇒ Cost function: (cost of parameters)

(Average of loss functions over a entire training set)

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

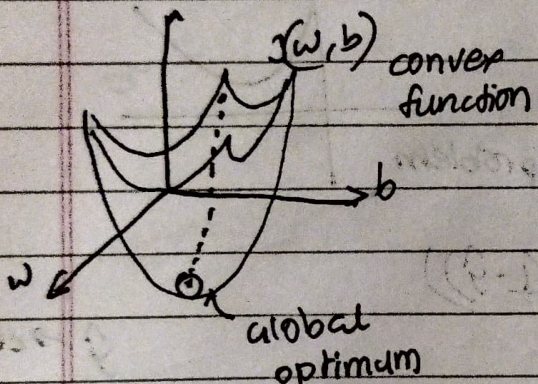
$$= -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

Gradient Descent

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

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

want to find w, b such that minimize $J(w,b)$



Formula

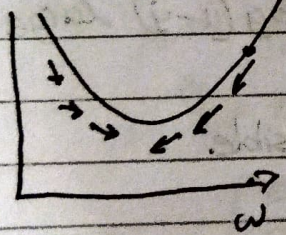
Repeat {

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

(updates)

"dw"

}



In 3d

$$\{ \quad w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

}

Derivatives

Kya Dekhu

Boring

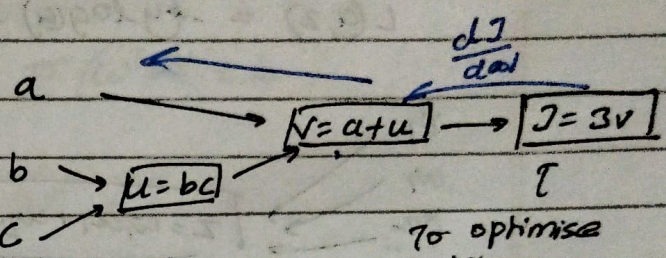
Computation Graph

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

$u = bc$

$v = a + u$

$J = 3v$



Derivatives with a computation graph

$J = 3v$

$$\frac{dJ}{dv} = 3 \quad \therefore \text{"}dv\text{"} = 3$$

$$\frac{dJ}{da} = \frac{dv}{da}\frac{dJ}{dv} \quad \therefore \text{"}da\text{"} = 3$$

$$= 3 \times 1 = 3$$

Final output variable J, many variables & with

$$\frac{d\,\text{Final output var}}{d\,\text{var}} \rightarrow \text{"}d\text{var}\text{"}$$

similarly $du = 3$

$db = 6$

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

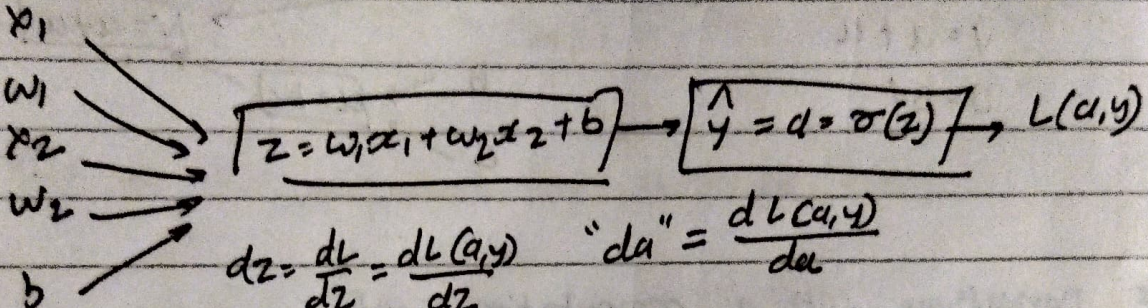
3 2

Logistic Regression Gradient Descent

$$z = w^T x + b$$

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

$$L(a, z) = -(y \log(a) + (1-y) \log(1-a))$$



$$dz = \frac{dL}{dz} = \frac{dL(a,y)}{dz} \quad \text{"}da\text{"} = \frac{dL(a,y)}{da}$$

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

$$= a - y \qquad a(1-a)$$

$$= \boxed{\frac{dL}{da}} \boxed{\frac{da}{dz}}$$

$$\frac{dL}{dw_1} = \text{"}dw_1\text{"} = x_1 dz \qquad dw_2 = x_2 dz \qquad db = dz$$

$$w_1 = w_1 - \alpha dw_1$$

$$w_2 = w_2 - \alpha dw_2$$

$$b = b - \alpha db$$

LATION

$$\left(\frac{1}{1+e^{-z}}\right)^2$$

$$\frac{e^{-z}}{(1+e^{-z})^2} \qquad \left(\frac{2}{1+e^{-z}}\right)$$

$$-\frac{(1+e^{-z})+1}{(1+e^{-z})^2} = -\frac{1}{1+e^{-z}} + \frac{1}{(1+e^{-z})^2}$$

$$= -a + a^2 \qquad a(a-1)$$

$$\frac{1+e^{-z} + 2(e^{-z})}{(1+e^{-z})^2}$$

$$= \frac{1}{1+e^{-z}} + \frac{2e^{-z}}{(1+e^{-z})^2}$$

☆ Logistic Regression on m examples

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

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

$$\frac{\partial}{\partial w_i} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w} L(a^{(i)}, y^{(i)})}_{dw^{(i)} = (a^{(i)}, y^{(i)})}$$

Initialise $J=0;\ dw_1=0;\ dw_2=0;\ db=0$

For $i=1$ to m

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

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

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

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

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

$dw_2 += x_2^{(i)} dz^{(i)}$ } $n=2$

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

$J /= m$

$dw_1 /= m;\ dw_2 /= m;\ db /= m$

$dw_1 = \frac{dJ}{dw_1}$

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

Vectorization

Art of getting rid of explicit for loop in code

$z = w^T x + b$

$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $w \in \mathbb{R}^{n_x}$ $z \in \mathbb{R}^{n_x}$

Non vectorized:

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

} than this

vectorized:

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

} Takes much less time

GPU / CPU } SIMD — single instruction multiple data

⭐ whenever possible, avoid explicit for loop

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

$u = Av$

$u_i = \sum_j A_{ij} v_j$

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

```
u = np.dot(A,v)
np.log(v)
np.abs(v)
np.maximum(v,0)
v**2
1/v
```

Instead of $dw1 = 0$, $dw2 = 0$

$dw = np.zeros((n_x, 1))$

One for loop removed

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

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

$\Rightarrow \boxed{dw += x^{(i)} dz^{(i)}}$ vectors

$dw_1 = \frac{dw_1}{m}$ $\frac{dw_2}{d} = \frac{dw_2}{m}$ $\Rightarrow \boxed{dw /= m}$

Vectorizing Logic Regression

$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(2)} = w^T x^{(2)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(2)} = \sigma(z^{(2)})$

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$

$R^{(n_x, m)}$

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

$1 \times m$

$$w^T \begin{bmatrix} \end{bmatrix} = [w^T x^{(1)} + w^T x_2 + w^T x_3 + \cdots] \; [b \; b \; b$$

$$= [w^T x^{(1)} + b + w^T x^{(2)} + b + w^T x^{(3)} + b + \cdots]$$

$\Rightarrow \boxed{Z = np.dot(w.T, X) + b}$ → Broadcasting takes place & this becomes $[b \; b \; \cdots \; b]$

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

$dz^{(1)} = a^{(1)} - y^{(1)}$ $dz^{(2)} = a^{(2)} - y^{(2)}$

$dz = [dz^{(1)}\ dz^{(2)} \dots dz^{(m)}]$

$A = [a^{(1)} \dots a^{(m)}]$ $Y = [y^{(1)} \dots y^{(m)}]$

$dz = A - Y = [a^{(1)} - y^{(1)}\ \ a^{(2)} - y^{(2)} \dots]$

$dw = 0$

$db = 0$

$dw \mathrel{+}= x^{(1)} dz^{(1)}$
$dw \mathrel{+}= x^{(2)} dz^{(2)}$
$\vdots$
$dw /= m$

$db \mathrel{+}= dz^{(1)}$
$db \mathrel{+}= dz^{(2)}$
$\vdots$
$db /= m$

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

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

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

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

$$= \frac{1}{m} [x^{(1)} dz^{(1)} + \dots + x^{(m)} dz^{(m)}]$$

$n \times 1$

Implementing using vectors

$Z = w^T X + b$
$= np.dot(w^T, X) + b$
$A = \sigma(Z)$
$dZ = A - Y$
$dw = \frac{1}{m} X dZ^T$
$db = \frac{1}{m} np.sum(dZ)$
$w := w - \alpha dw$
$b := b - \alpha db$

For iteration of whole for loop req.

Broadcasting in Python

	Apples	Beef	Eggs	Potatoes
Carb	56	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= A_{(3,4)}$

calculate % of calories from c, p, F.
can you do this without for loop?

```
cal = A.sum(axis=0)  ← sum vertically
print(cal)
[59, 239, 155.4, 76.9]
```

(?) → percentage = 100 * A / cal.reshape(1, 4)

↑ (3,4) / (1,4) [used to reshape matrix

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + 100 \Rightarrow \begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix}$$

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

$$\Downarrow$$

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

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

$$\Downarrow$$

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

General Principle

(m,n) matrix + − * /

(1,n) → (m,n)

(m,1) → (m,n)

(m,1) + ℝ

$\begin{bmatrix}1\\2\\3\end{bmatrix}$ + 100 = $\begin{bmatrix}101\\102\\103\end{bmatrix}$

Matlab/octave: bsxfun ← broadcasting

Python / Numpy

Tips & tricks to eliminate bugs

Specify rows and columns always as this may create a ambiguity blw array & matrix.

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

} Beaware Don't use

```
a = np.random.randn(5,1)   → column vector
a = np.random.randn(1,5)   → Row vector
```

```
assert (a.shape == (5,1))  ←
a = a.reshape((5,1))  ← to convert as per need
```

Jupyter / iPython Notebooks

→ Shift + Enter to Execute program

→ Restart ∞ kernel if get any errors

→ submit assignment

Explanation of Logistic Regression cost Function

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

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

If $y=1$: $P(y/x) = \hat{y}$

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

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

$$\log P(y/x) = y \log \hat{y} + (1-y) \log (1-\hat{y})$$

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

cost on m examples

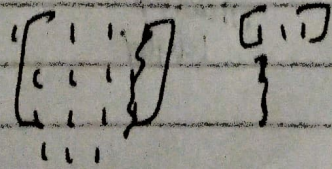
$$\log P(\text{labels in training set}) = \log \prod_{i=1}^{m} P(y^{(i)}/x^{(i)})$$

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

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

Maximum likelihood estimation

$$\text{cost } J(w,b) = -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$



0.5 0.1